Ville Perälä

Hybridimobiilisovelluksen kehittäminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

18.4.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Ville Perälä Hybridimobiilisovelluksen kehittäminen 41 sivua 18.4.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Projektivastaava Lasse Ilvessalo Lehtori Juha Kämäri

Opinnäytetyössä tarkastellaan hybridimobiilisovelluksen kehittämistä teknologian ja käytännön näkökulmista. Lisäksi esitellään ja vertaillaan kehittämisen muita vaihtoehtoja sekä esitetään valintakriteereitä kehitystavan valintaan.

Työssä käydään läpi mobiilisovelluksen kehitysprosessi käyttäen Ionic Framework 2 -sovelluskehystä. Samalla käydään läpi, miten hybridisovellus toimii mobiililaitteessa ja mitä sovelluskehys tarjoaa kehityksen tueksi. Konkreettisena osana työtä rakennetaan Intellia Oy:lle hakupalveluita tarjoava sovellus työssä esiteltyjen tekniikoiden avulla.

Kehitysprosessista käydään läpi sovelluskehyksen tarjoamia käyttöliittymäkomponentteja, sovelluksen rakennetta ja kehyksessä käytettäviä suunnittelumalleja. Lisäksi esitellään kehyksen käyttöön liittyviä kehitystyökaluja.

Opinnäytetyön lopputuloksena syntyi monipuolinen yritys- ja yhteystietojen hakusovellus, joka toimii Android- ja iOS-käyttöjärjestelmillä. Sovelluksen kehityksen kuvauksessa otetaan kantaa niin teknisiin toteutusvaihtoehtoihin kuin käyttöliittymän käytettävyyteenkin. Kokonaisuutena opinnäytetyöstä muodostuu kattava läpileikkaus mobiilisovelluskehityksen eri näkökulmiin teoriatasolta käytännön yksityiskohtiin.

Avainsanat	Hybridimobiilisovellus, JavaScript, HTML5, Ionic Framework, Angular

Author Title Number of Pages Date	Ville Perälä Development of Hybrid Mobile Application 41 pages 18 April 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Lasse Ilvessalo, Project Manager Juha Kämäri, Lecturer

This thesis studies the development of hybrid mobile applications from both technological and practical points of view. Other options for mobile development are also reviewed and a selection criteria for selecting a fitting development method are introduced with the reviews.

The development process for a hybrid mobile application is described using the Ionic Framework 2 development framework. The inner functionality of a hybrid application is also discussed and the supporting features of a development framework are highlighted. The practical part of this thesis is about making a mobile search application for Intellia Oy with the aforementioned technologies.

User interface components, structure of an application and the design patterns of the framework are reviewed thoroughly in the development process. In addition, other related development tools are also introduced.

The result of this project is a versatile mobile application for searching company and contact information working in both Android and iOS operating systems. Both technical aspects and user experience are key points in the description of making the application. With the theoretical and practical perspectives of this thesis, it becomes a comprehensive overview of the different aspects of current mobile software development.

Keywords	Hybrid mobile application, JavaScript, HTML5, Ionic Framework, Angular

Sisällys

Lyhenteet

CSS	Cascading Style Sheets. Tyyliohjeiden määrittelyyn käytettävä kieli, jolla kuvaillaan, miltä dokumentin tulee näyttää, kun se esitetään eri medioissa.
HTML5	HyperText Markup Language versio 5. Kuvaileva merkkauskieli, jonka avulla kuvataan verkkosivun rakennetta. Viides versio laajentaa standardia tuoden mukanaan aiempaa laajemman tuen muun muassa multimedialle ja tiedon tallennukselle. Lisäksi mukana on rajapintatuki päätelaitteen fyysisille ominaisuuksille, kuten sijaintitietoon ja kameraan.
IDE	Integrated Development Environment. Sovelluskehitysympäristö, jossa vähintään koodin kääntäjä on sovitettu yhteen koodieditorin kanssa.
JSON	JavaScript Object Notation. Tekstipohjainen JavaScript-olioiden tai muun vastaavasti jäsennellyn tiedon kevyt tallennusmuoto.
MV*	Model View *. Ohjelmiston suunnittelumalli, jossa hyödynnetään näkymiin sidottuja malleja. Lyhenteen tähtimerkinnällä tarkoitetaan näkymien ja mallien sitovaa osioita, joka voi olla esimerkiksi sovelluskehyksestä riippuen kontrolleri, ViewModel-komponentti, näiden risteytys tai jotain muuta.
NPM	Node Package Manager. Pakettienhallintajärjestelmä Node.js-pohjaisille sovelluksille.
REST	Representational State Transfer. REST on http-protokollaan perustuva arkkitehtuurimalli, jota käytetään tyypillisesti rajapintojen rakentamiseen asiakkaan ja palvelimen välille.
SASS	Syntactically Awesome Style Sheets. Tyylien määrittelyyn käytettävä ohjelmointikieli, joka kääntyy CSS:ksi. Kieli mahdollistaa muun muassa muuttujien ja toistorakenteiden hyödyntämisen tyylien määrittelyssä.
URI	Uniform Resource Identifier. Merkkijono, jolla esitetään tietyn resurssin sisällön osoite ja valinnaisesti sen käyttämä protokolla tai skeema.

URL	Uniform Resource Locator. Merkkijono, joka sisältää halutun resurssin web-osoitteen sekä sen hakemiseen käytettävän protokollan, esim. http://www.example.com. URL on yleisin muoto URI:sta.

1 Johdanto

Mobiililaitteiden räjähdysmäisen kasvun on jokainen voinut todeta viime vuosien aikana. Siihen liittyy päätelaitteiden ominaisuuksien parantumisen lisäksi olennaisena osana sovellustarjonnan kasvu ja monipuolisuus eri laitealustoilla. Sovelluskehittäjien tueksi onkin kehitetty erilaisia vaihtoehtoisia tapoja rakentaa mobiililaitteilla toimivia sovelluksia, jolla jokaisella on omat vahvuutensa ja rajoitteensa.

Tässä opinnäytetyössä keskitytään mobiililaitteiden sovelluskehitykseen ja erityisesti niiden kehittämiseen yhdistäen laitealustan tarjoamat omat rajapinnat sekä moderni verkkosovelluskehitys. Työssä vertaillaan mobiilisovelluskehityksen eri vaihtoehtoja sekä perustellaan kehitystavan valintaa suhteessa kehitystyön lähtökohtiin, rajoitteisiin ja vaatimuksiin. Lisäksi esitellään ja käydään tarkemmin läpi Ionic Framework -hybridimobiilisovelluskehyksen toimintaa ja hybridimobiilisovellusten rakentamista aina käyttöliittymäkomponenteista sovelluslogiikkaan.

Työn käytännön osuus käsittelee asiakastyönä tehtyä mobiilisovellusta Intellia Oy:lle, jonka tavoitteena on yhdistää Intellian tarjoamat yritys- ja henkilöhakupalvelut mobiililaitteella toimivaksi kokonaisuudeksi. Tavoitteena on lisäksi integroida sovellus mobiililaitteen natiiveihin toimintoihin. Sovellus toteutetaan Ionic Framework -sovelluskehyksen uusimmalla, vielä kehitysvaiheessa olevalla versiolla.

Opinnäytetyöstä muodostuu kokonaisuutena kattava katsaus tämän hetken mobiilisovelluskehityksen eri vaihtoehtoihin sekä tarkempi selvitys hybridimobiilisovellusten rakentamisesta Ionic Frameworkilla. Lisäksi mobiilisovelluskehityksen eri näkökulmat konkretisoituvat Intellian mobiilisovellusprojektin muodossa, missä otetaan teknisten ratkaisujen lisäksi kantaa sovelluksen käytettävyyteen ja ulkoasuun.

2 Mobiilisovelluksen kehittämisen vaihtoehdot

Mobiilisovelluksien kehitys jakautuu kolmeen eri lähestymistapaan: natiivisovelluksiin, verkkosovelluksiin sekä hybridisovelluksiin (1). Näiden lisäksi on olemassa eri kehitysvaihtoehtojen risteytyksiä, jotka on havainnollistettu kuviossa 1.

Kuvio 1. Mobiilisovelluksen kehittämisen eri vaihtoehtojen karkea jakautuminen

Kehitysvaihtoehtojen pääkategorioiden lisäksi on olemassa erilaisia tapoja toteuttaa sovellus kehitysvaihtoehdolla. Verkkosovelluksen voi tehdä esimerkiksi joko perinteisenä verkkosivuna tai laitteeseen ladattavana verkkosivuna, kuitenkin ilman natiiveja toiminnallisuuksia. Hybridimobiilikehityksellä voidaan tarkoittaa joko kokonaisen verkkosovelluksen suorittamista asennettavana sovelluksena kohdelaitteessa tai verkkosisällön käyttämistä natiivien toiminnallisuuksien lisänä sovelluksessa (2).

Jokaisella mobiilikehitysvaihtoehdolla on omat vahvuutensa ja heikkoutensa sekä esimerkkejä niillä luoduista sovelluksista. Eri kehitysvaihtoehtojen vertailussa ei olekaan kyse eri vaihtoehtojen asettamisesta paremmuusjärjestykseen vaan siitä, miten eri sovelluskehitykseen liittyvät vaatimukset ja rajoitteet saadaan tasapainotettua mahdollisimman hyvin lähtökohdat huomioon ottaen. Tässä luvussa käsitellään eri lähestymistapojen tarkastelemisen lisäksi niiden etuja ja haittoja koko mobiilisovelluskehitysprosessin aikana.

2.1 Natiivi mobiilisovellus

Pcmag määrittelee yksinkertaisesti natiivin mobiilisovelluksen olevan sovellus, joka on ohjelmoitu käyttöjärjestelmälle suositeltavalla ohjelmointikielellä. Esimerkiksi Googlen Androidilla tämä tarkoittaa Javaa ja Applen iOS:llä Objective-C:tä tai Swiftiä. (3.)

Natiivi mobiilisovellus tarjoaa kehittäjälle lähtökohtaisesti pääsyn kaikkiin laitteen ominaisuuksiin ja rajapintoihin sekä on suorituskyvyltään nopein vaihtoehto. Esimerkiksi raskaissa mobiilipeleissä graafisen suorituskyvyn tehokas hyödyntäminen ja mahdollisimman pienen latenssin aikaansaaminen eivät onnistu selainpohjaisilla ratkaisuilla tarpeeksi hyvin, joten mobiilipelejä kehitettäessä natiivi sovellus on lähes ehdoton vaihtoehto. (1.)

Käyttäjälle natiivisovellus erottuu suorituskyvyn lisäksi käyttöliittymäkomponenttien saumattomuudessa muun käyttöjärjestelmän kanssa. Sovelluksista saadaan pienellä vaivalla yhdenmukaisia ja paremmin integroituvia käyttöjärjestelmään kuin verkkopohjaisilla sovelluksilla, joissa vastaavan käyttökokemuksen saaminen vaatii enemmän työtä.

Natiivien sovellusten julkaisukanavina toimivat erilaiset sovelluskaupat, esimerkiksi Googlen Play-kauppa ja Applen App Store. Sovelluksen julkaisuprosessi sovelluskauppaan on melko suoraviivainen, mutta maksullinen prosessi (4; 5). Esimerkiksi App Store veloittaa kehittäjälisenssin hinnan lisäksi 30 % sovelluksen myynnistä sekä sovelluksen sisäisistä ostoista (4). Lisäksi ladattava sovellus vie aina tilaa käyttäjän laitteesta, mikä voi vaikuttaa käyttäjän lataus- tai ostopäätökseen (1). Esimerkiksi, mikäli käyttäjä tarvitsee juna-aikataulutietoja harvoin, hän todennäköisemmin katsoo ne mieluummin verkkosivulta tarpeen mukaan kuin säilyttää aikataulusovellusta puhelimessaan.

Sovelluksen julkaisijan omien taustapalveluiden kattavuus vaikuttaa lisäksi natiivin sovelluksen kehittämispäätökseen. Jotta natiivisovellukseen saadaan päivitettyä tietoa sekä esimerkiksi käyttäjien välistä interaktiota, tarvitaan tukea taustapalveluilta. Usein tuki tarjotaan erilaisina rajapintoina, joiden täytyy mukautua mobiilisovelluksen tarpeisiin. Kinveyn ja MooWebin kirjassa mainitaankin, että joissain tapauksissa rajapinnan luominen täysin natiiville yrityssovellukselle voi olla suurempi työmäärältään kuin itse sovelluksen rakentaminen. (1.)

2.2 Verkkosovellus

Verkkosovelluksen erottamiseksi pelkästä verkkosivusta ei ole selkeää rajaa. Eron voi kuitenkin huomata esimerkiksi siitä, että verkkosivu keskittyy tiedon jakamiseen ja näyttämiseen verkkosovelluksen keskittyessä käyttäjän ja sovelluksen väliseen interaktioon ja tiedon käsittelyyn.

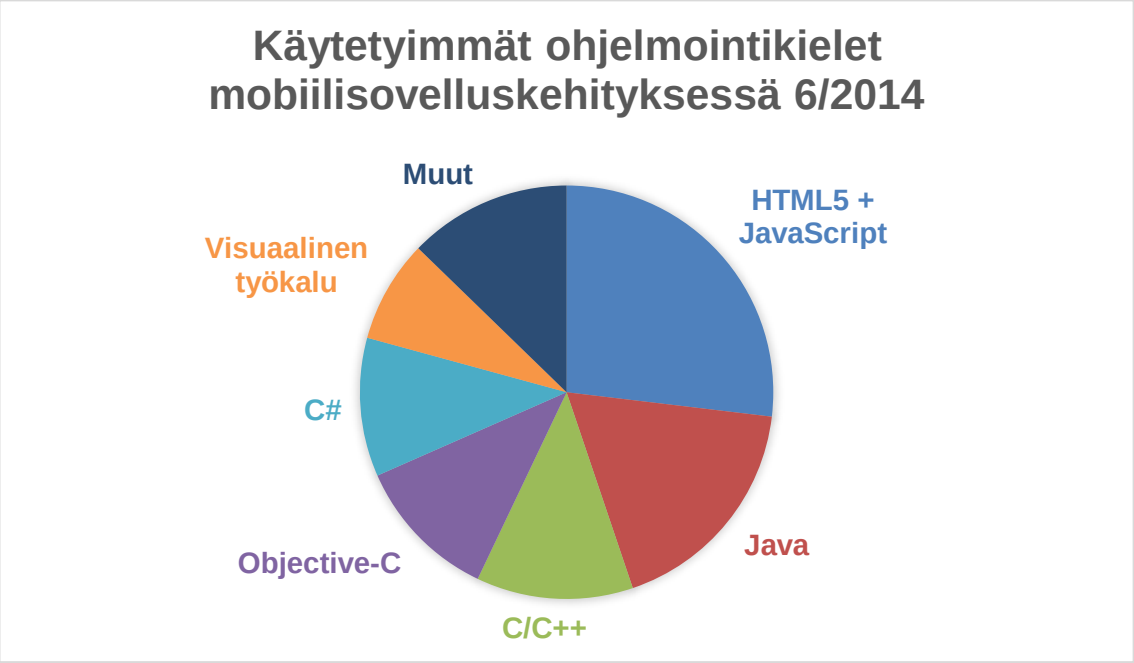
Mikäli sovellus ei tarvitse pääsyä laitteen natiiveihin toimintoihin, esimerkiksi kameraan tai eri sensoreihin, eikä sovellusta haluta julkaista erillisessä sovelluskaupassa, verkkosovelluksen rakentaminen muodostuu kustannustehokkaaksi ratkaisuksi. Koodipohjan ylläpito ja päivitysten julkaisu on myös helppoa ja nopeaa ilman useita eri versioita samasta sovelluksesta eri alustoille. (1.)

Kuitenkaan suurin osa käyttäjäkunnasta ei käytä verkkosovelluksia vaan puhelimeen asennettuja sovelluksia. 85 % älypuhelinten käyttäjien ajasta kuluu asennettuja sovelluksia käytettäessä Forresterin tutkimuksen mukaan, ja määrä on noussut yli 60 % kahden vuoden aikana (6; 7). Lisäksi Nielsenin tutkimuksen mukaan kuluttajat käyttävät keskimäärin 26,7 sovellusta kuukausittain (7). Näiden tilastojen valossa verkkosovelluksen valinnaksi kehitysalustaksi vaatii käyttäjätutkimuksen kohderyhmästä, jotta voidaan varmistua siitä, että sovellukselle riittää käyttäjiä.

2.3 Hybridimobiilisovellus

Hybridimobiilisovelluksella tarkoitetaan kahden aiemman esitetyn mobiilisovelluskehitysvaihtoehdon yhteenliittymää; rakennetaan natiiveja mobiililaitteen ominaisuuksia hyödyntävä web-sovellus, joka asennetaan mobiililaitteeseen. Tämä toteutetaan usein luomalla sovellus, joka käynnistyessään käynnistää joko alustan verkkoselaimen koko näytön tilassa ja suorittaa verkkosovelluksen tai hyödyntää alemman tason web-sisällön näyttämiseen käytettävää rajapintaa (1). Mobiilisovelluksesta itsestään on vaikea päätellä, millä tekniikalla se on rakennettu, ja sovelluksen julkaisijat eivät usein jaa tietoa ollenkaan. Kuitenkin tutkimalla tämän hetken suurien käyttäjämäärien sovelluksien teknologiablogeja voidaan päätellä, että esimerkiksi Netflix ja LinkedIn on rakennettu hybriditekniikoilla (8; 9). Lisäksi suositun opetusalustan Moodlen mobiiliversio on myös rakennettu hybriditekniikoilla (10.)

Mobiilisovelluskehittäjät tuottavat sovelluksia hybriditekniikoilla paljon, mikä näkyy esimerkiksi VisionMobilen 2014 kesäkuussa julkaisemassa laajassa kehittäjätutkimuksessa (11). Kuvioon 2 on koostettu kehittäjien eniten käyttämät ohjelmointikielet mobiilisovelluskehityksessä.



Kuvio 2. Ohjelmointikielien jakautuminen mobiilisovelluskehityksessä kesäkuussa 2014. Kooste VisionMobilen tutkimuksesta (11)

Kuviossa perinteisten olio-ohjelmointikielien kuten C:n, C#:n, Objective-C:n ja Javan joukossa on lisäksi HTML5 (HyperText Markup Language versio 5), eli verkkodokumenttien rakennetta kuvaava merkkauskieli. Tähän yhdistetään yleensä CSS:n (Cascading Style Sheets) käyttö, joka tuo mahdollistaa merkkauskielen eri osien vapaan tyylittelyn eri tiedostoista. Varsinainen ohjelmointikieli HTML5 ei ole, vaan sen toiminnallisuus tulee JavaScript-ohjelmointikielen käytöstä HTML-dokumenttien rinnalla.

Kuviossa eniten käytetty kieli on 57 prosentin osuudella HTML5- ja JavaScript-teknologiapino, mihin sisältyy myös kaikki JavaScriptiksi käännettävät kielet kuten Coffee- ja TypeScript. Tämä kertoo verkkokehitystekniikoiden suosiosta. Käytännössä suurin osa kehittäjistä käyttää verkkosovellustekniikoita mobiilikehityksessä jollain tapaa. Luku ei kuitenkaan kerro pelkästään hybridisovellustekniikoiden käytöstä, sillä tekniikoilla on mahdollista tuottaa myös pelkkiä verkkosovelluksia. Trendin ollessa kasvusuuntainen, voidaan päätellä, että tulevaisuudessa verkko- ja hybriditekniikoilla tehdään enemmän

mobiilisovelluksia kuin perinteisillä natiivitekniikoilla, ainakin käyttöjärjestelmäkohtaisesti.

VisionMobilen tutkimuksessa mainitaan lisäksi, että 47 % iOS kehittäjistä ja 42 % Android-kehittäjistä käyttää jotain muuta kuin alustansa natiivia ohjelmointikieltä. HTML5- ja JavaScript-teknologioiden ollessa käytetyin ohjelmointiteknologia, suuri osa kehittäjistä vaikuttaisi käyttävän verkkosovelluksen kehitysmenetelmiä työssään. Kuitenkin pääasialliseksi kehityskieleksi HTML5 ja JavaScript yltää vain alle 20 %:lla kehittäjistä, jolloin vaikuttaisi siltä, että hybridi- ja verkkosovellustekniikoita käytetään paljon, mutta ei pääasiallisena sovelluksien kehitystapoina. Suunta tulee todennäköisesti olemaan kasvava teknologioiden kehittyessä ja järjestelmätason verkkosovellustekniikoiden tuen parantuessa. Esimerkiksi Google on ottanut käyttöön suoran tuen laitteisiin asennettaville verkkosovelluksille Android-käyttöjärjestelmälleen. (12; 11.)

2.4 Kuinka valita mobiilisovelluksen kehitysmuoto?

Ennen mobiilisovelluksen kehityspolun valintaa on otettava huomioon kehitystyöhön käytettävät resurssit ja lopputuotteeseen kohdistuvat vaatimukset, kuten aikamääreet, osaamistaso, budjetti sekä kohdeyleisö. Esimerkiksi seuraavan kaltainen lista kehityksen taustoihin liittyvistä asioista selkeyttää valintaprosessia:

- Mille alustoille sovellus tulee julkaista?
- Mikä on kehityksen aikataulu ja/tai budjetti?
- Tarvitaanko sovellukselle mahdollisimman suuri suorituskyky kohdelaitteelta?
- Onko sovellukselle olemassa jo soveltuvia taustapalveluita?
- Riittäkö sovelluksen kehittäjän/kehittäjien tietotaito oman sovelluksen rakentamiseksi jokaiselle kohdealustalle?
- Tarvitseeko sovellus päätelaitteen natiiveja toimintoja (esim. yhteystietoluettelo, sensorit)?

- Millainen on sovelluksen kohdeyleisö? Haluavatko he mieluummin sovelluskaupoista ladattavan sovelluksen vai nopeasti vierailtavan verkkosivun?

Listan eri kohtien merkitykset on hyvä käydä läpi konkreettisen sovellusprojektin avulla. Työn käytännön osuutta varten rakennettavassa Intellian mobiilisovelluksessa kehitysvaihtoehdon valinta hybridimobiilisovellukseksi osoittautui verrattain suoraviivaiseksi.

Sovellus tarvitsee natiiveja toiminnallisuuksia, kuten tuen yhteystietojen luomiselle päätelaitteeseen, jolloin pelkkä verkkosovellus ei riitä vaatimusten täyttämiseen. Samalla kuitenkin alustatuki halutaan pitää mahdollisimman laajana, mutta aikataulu ei riitä erillisiin natiiveihin sovellusratkaisuihin. Lisäksi sovellukselle on jo olemassa verkkopalveluille optimoidut rajapinnat ja verkkosovellustekniikat ovat entuudestaan tuttuja, jolloin hybridimobiilisovellus muodostuu järkevimmäksi vaihtoehdoksi (1.)

Hybridimobiilisovelluksen rakentamisessa käytetään usein apuna sovelluskehystä, mikä tehostaa ja helpottaa koko kehitysprosessia. Erilaisten työkalujen ja kehityskirjastojen sekä sovelluskehysten määrän ollessa valtava, sovelluskehyksen valintaan täytyy kiinnittää huomioita. Hyvä ja toimiva sovelluskehys nopeuttaa kehitystyötä, tekee koodipohjasta helposti ylläpidettävää ja laajennettavaa sekä mahdollistaa luotujen komponenttien ja sovellusosien uudelleenkäytön muissa saman sovelluskehyksen projekteissa. Intellian mobiilisovelluksen kehitys päädyttiin toteuttamaan Ionic Framework 2 -sovelluskehyksellä. Valinnan perusteet esitellään luvussa 3.3.

3 Ionic Framework

Ionic Framework on HTML5-pohjainen kehys mobiilisovelluksien luomista varten. Sen kehittäjänä toimii Drifty Co., ja se on rakennettu erityisesti hybridimobiilisovelluksien rakentamista varten. Ionic Framework ei ole tarkoitettu toimimaan verkkosovellusten tavoin laitteen selaimessa, esimerkiksi Applen Safarilla tai Googlen Chromella, vaan suoraan laitteen alimman tason selainnäkymässä tarjoten valmiit käyttöliittymäkomponentit ja sovellusrakenteen sekä muita toiminnallisuuksia. Ionic korostaa myös verkkosivuillaan, että kehyksen rakentamisen tavoitteena on luoda täysin avoimen lähdekoodin toteuttava ympäristö, jonka ympärille rakennetaan lisäksi vahva kehittäjäyhteisö. (13).

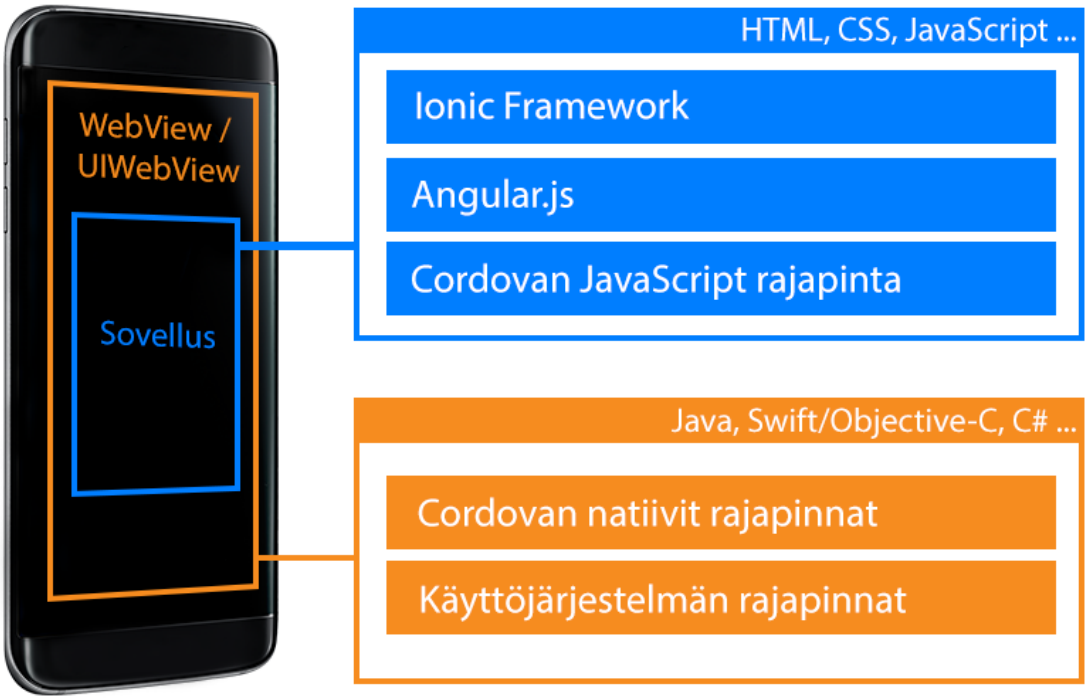
3.1 Miten sovelluskehys toimii?

Ionic käyttää perustanaan käyttöjärjestelmän alemman tason selainnäkymää, Androidissa tämä on WebView ja iOS:ssä UIWebView. Selainnäkymät kykenevät näyttämään verkkosovelluksia natiivin sovelluksen sisällä, mikä mahdollistaa hybridisovellusten rakentamisen. (13.)

Selainnäkymistä on verkkosisällön näyttämisen lisäksi pääsy laitetason toimintoihin laitteen omien rajapintojen kautta. Näitä natiiveja toiminnallisuuksia sidotaan Ionic Framework -kehykseen Cordovan avulla. Cordova on oma sovelluskehyksensä, joka paketoi kaikkien alustojen natiivit rajapintakäskyt toimimaan suoraan alustan selainnäkymässä ja tarjoaa lisäksi JavaScript-kirjaston näiden kutsumiseen. Cordova ei kuitenkaan sisällä käyttöliittymäkomponentteja, eikä ole sidottu mihinkään erityiseen sovelluskehykseen. (14; 15.)

Ionic tuo Cordovan päälle valmiit natiivisovelluksien kaltaiset käyttöliittymäkomponentit ja lisäksi paketoi kaikki yksittäiset teknologiat yhtenäiseksi kokonaisuudeksi. Ionic Framework on sovelluskehitysalustana rakennettu Angular.js-sovelluskehyksen pohjalle. Angular on Googlen rakentama ja ylläpitämä JavaScript-sovelluskehys, joka perustuu MV*-arkkitehtuuriin. MV*:llä tarkoitetaan Model-View-sovellusarkkitehtuuria, jossa kuitenkin näitä kahta sitova osa on vapaavalintainen, se voi olla esimerkiksi ViewModel, Controller tai jotain muuta (13.)

Ionic pyrkii ratkaisullaan yhdenmukaistamaan mahdollisimman paljon eri laitteistojen eroja sovelluslogiikasta, jotta kehittäjän ei tarvitse välttämättä luoda erilaista ohjelmakoodia eri alustoille. Alustakohtaista sovelluslogiikkaa ja ulkoasujen määrittelyä voidaan kuitenkin tehdä tarpeen mukaan (11). Kuviossa 3 on havainnollistettu Ionic-kehyksen koko teknologiapino. (13; 14; 15.)

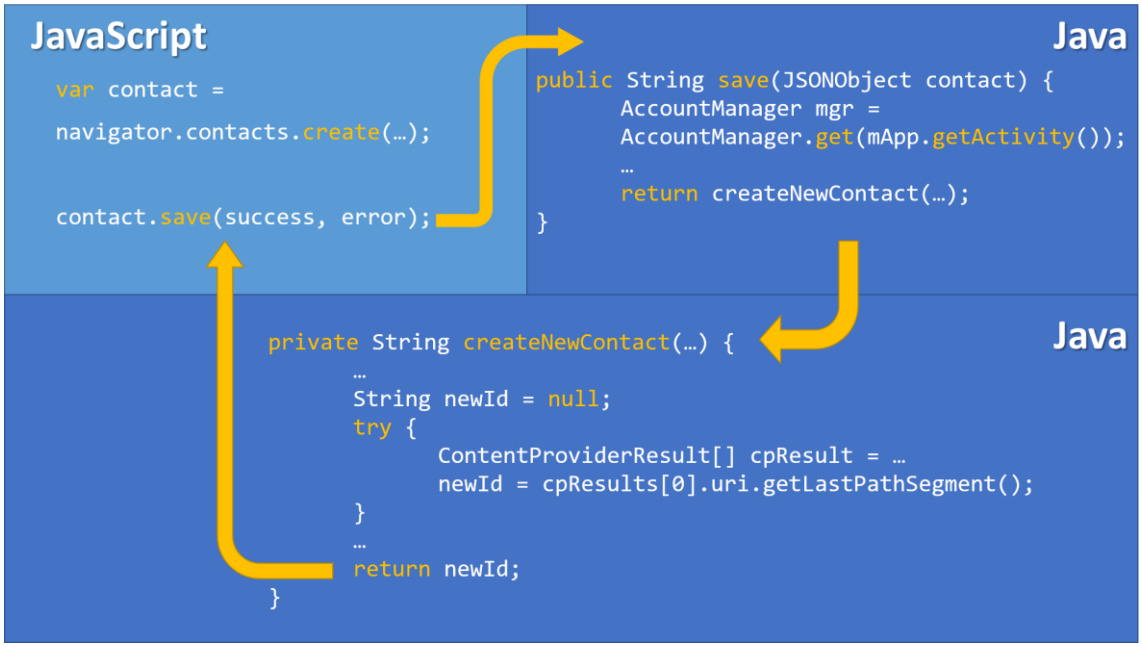


Kuvio 3. Ionic Framework -sovelluskehyksen teknologiapino havainnollistettuna

Kuviossa on eritelty Ionic-sovelluksen käyttämät teknologiat ja sovelluksen suorittamiseen tarvittavat teknologiat omiksi laatikoikseen ylhäältä alaspäin. Itse sovellus rakennetaan Ionic-kehyksen ja Angularin avulla verkkosovelluksena. Koko teknologiapinon kulmakivenä on kuitenkin Cordovan toiminta käyttöliittymän ja natiivin toiminnallisuuden siltaajana, mikä mahdollistaa järjestelmäriippumattoman sovelluskehityksen.

3.2 Cordovan toiminta

Kun Ionic-sovellus tarvitsee natiiveja toiminnallisuuksia, esimerkiksi yhteystietoluetteloa, kutsutaan toimintoa Cordovan JavaScript-rajapinnan avulla. Cordova ohjaa pyynnön laitteen käyttöjärjestelmän mukaan omaan natiiviin toteutukseensa, joka suorittaa käyttöjärjestelmän rajapintoja hyödyntäen halutun toiminnon. Kutsun paluuarvo välitetään takaisin kerroksien läpi sovelluksen sisään. Monikerrosarkkitehtuurin luonteesta johtuen rajapintakutsut toteutetaan tyypillisesti asynkronisesti eli irrallisena runkosovelluksen toiminnasta, jotta natiivien toiminnallisuuksien kutsuminen ei aiheuta runkosovelluksen toiminnan pysähtymistä kutsujen ajaksi. Kuviossa 4 on havainnollistettuna tarkemmin esimerkkinä toimivan yhteystietojen tallentamisen kutsupolku Ionic-sovelluksessa.



Kuvio 4. Havainnollistus yhteystietojen luomisen kutsupolusta Android-käyttöjärjestelmällä

Kuvassa vasemmassa yläkulmassa sovellus kutsuu natiivia yhteystietojen luomisen metodia contacts.create, joka sijaitsee kaikkialla saatavissa olevassa navigator-oliossa. Useimmat Cordovan toiminnoista on sijoitettu navigator-olion sisälle, jolloin niitä on yksinkertaista käyttää, eivätkä ne ylikirjoita muita mahdollisesti samoin nimettyjä globaaleja toimintoja. Luotu yhteystieto yritetään tallentaa save-metodilla, joka lähettää kutsun Cordovan laajennustenhallinnan läpi vastaavaan Java-metodiin Androidissa. Mikäli kohdekäyttöjärjestelmä olisi iOS, kutsu reititettäisiin vastaavaan Objective-C-metodiin.

Julkinen save-metodi palauttaa kutsun yksityiseen createNewContact-metodiin, joka yrittää suorittaa yhteystiedon tallennuksen laitteen yhteystietoluetteloon. Tallennuksen tulos palautetaan takaisin JavaScript-kutsuun, joka määrittelee success- ja error-funktiot tallennuksen onnistumisen ja epäonnistumisen varalta. Cordova yhtenäistää näin eri laitealustat yhden laajennettavissa olevan rajapinnan alle.

3.3 Miksi Ionic Framework?

Kehitys erilaisissa hybridimobiilikehyksissä on ollut valtaisaa heti niiden ensiesiintymisestä lähtien. Esimerkiksi erilaisia mobiilikehyksiä kokoava Mobile Frameworks Comparison Chart sisältää kirjoitushetkellä 46 erilaista mobiilisovelluskehystä (16).

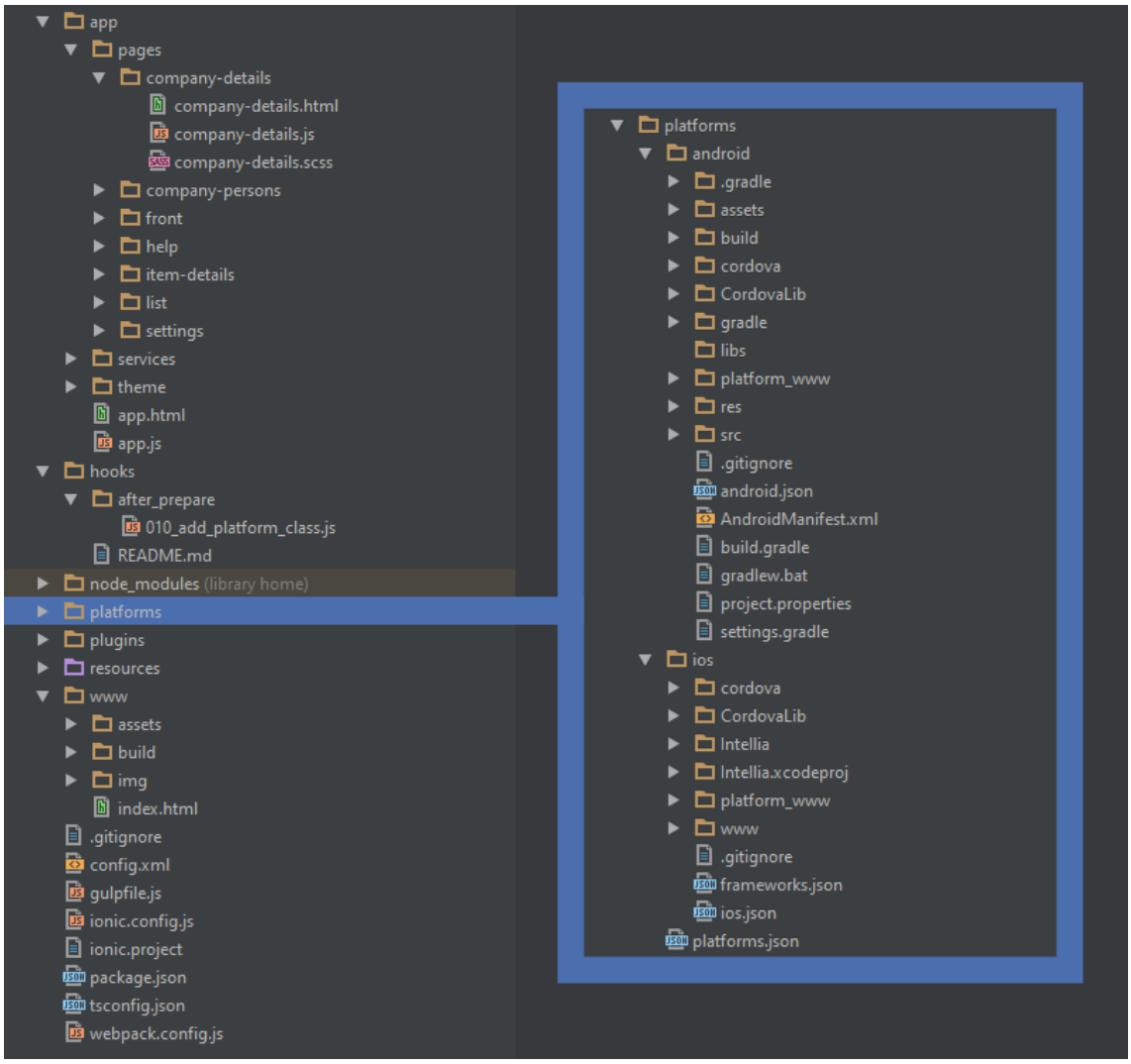
Ionic erottui edukseen muista vaihtoehtoisista sovelluskehyksistä sen kattavuudessa ja projektin aloittamisen nopeudessa ja helppoudessa. Moni muu kehys vaikuttaa olevan keskittynyt vain joihinkin yksittäisiin seikkoihin hybridisovelluskehityksessä sekä vaativat muita itse asennettavia lisäkirjastoja kokonaisuuden rakentamiseksi. Jotkin kehykset olivat myös muuten todella lupaavia, mutta aivan liian varhaisessa vaiheessa kehitystä tuotantokelpoisen sovelluksen rakentamiseksi.

Ionic Frameworkin dokumentaatio ja yhteisötuki ovat korkealla tasolla, jolloin kehitystyö on huomattavasti helpompaa kun kysymyksiin saa tarvittaessa vastauksia ja käytännön esimerkkejä löytyy paljon. Kehyksen avulla on lisäksi tehty lukuisia valmiita sovelluksia, joita tarkastelemalla kehyksen mahdollisuudet konkretisoituivat käytännössä. Kehykseen voi myös lisätä suoraan erilaisia Cordovan lisäosia, mikäli tarvitaan laitetason ominaisuuksia, joita itse kehyksestä ei suoraan löydy (11).

Edellä mainittujen seikkojen, kattavan dokumentaation sekä kehyksellä luotujen sovellusten määrän perusteella valitsin kirjaston Intellian mobiilisovelluksen pohjaksi. Lisäksi päädyin kehittämään sovellusta kehyksen uusimmalla beta-vaiheen versiolla, jotta sovelluskehitykseen voidaan tuoda mukaan JavaScript-kehityksen uusimpia ominaisuuksia ja parempi suorituskyky.

3.4 Projektirakenne

Ionic Framework tarjoaa erilaisia malliprojekteja, joista omaa sovellusta voi lähteä rakentamaan eteenpäin. Projektin tiedostorakenne on kuitenkin yhtenäinen projekteissa ja kehys tuottaa oletuksena tietynlaisen kansiorakenteen projektille kääntämisen jälkeen. Kansiorakenne antaa jo itsessään kuvan kehyksen toiminnasta, joten Intellian mobiilisovelluksen varhaisen kehitysvaiheen kansiorakenne näytetään kuviossa 5.



Kuvio 5. Projektin kehitysvaiheen kansiorakenne sovelluksen kääntämisen jälkeen

Kuviossa 5 olevien kansioiden sisällöt ja niihin liittyvät toiminnot on esitelty eriteltyinä seuraaviin alalukuihin.

3.4.1 Sovellus, hooks-kansio ja apukirjastot

Sovelluksen lähdekoodi löytyy kehyksessä app-kansion sisältä, jossa on usein ohjelmakoodin lisäksi myös tyylitiedostojen kääntämättömät SASS-tiedostot. SASS eli Syntactically Awesome Style Sheets on HTML tyylien kevyt ohjelmointikieli, joka kääntyy CSS-muotoiseksi. Se mahdollistaa esimerkiksi muuttujien ja toistorakenteiden hyödyntämisen tyylien määrittelyssä.

Sovellus jaetaan tyypillisesti käyttöliittymän eri sivuihin, jotka sisältävät käyttöliittymän komponentit. Kuviossa 5 komponentit löytyvät pages-kansion alta. Komponentit voivat sisältää sovelluslogiikan lisäksi myös HTML-pohjaisen näkymätiedoston, sekä moduulikohtaiset tyylitiedostot. (17.)

Cordovan toimintaan kehyksessä liittyvät kansiot plugins ja hooks. Plugins-kansio sisältää Cordovan laajennuskirjastot, esimerkiksi luvussa 3.2 esitelty yhteystietorajapinta löytyy tästä kansiosta. Hooks-kansio on Cordovan lisäominaisuus, joka mahdollistaa erilaisten esi- tai jälkikutsujen ajamisen sovelluksen kääntämisvaiheessa (18). Ionic kutsuu oletuksena esimerkiksi sovelluksen valmistelun jälkeen after_prepare-jälkikutsun ohjelmakoodia, jolla lisätään käännettävän alustan nimi sovelluksen HTML:n body-elementin luokkamääreeksi. Tällä tavalla sovellukseen saadaan sidottua erilaisia ulkoasuja helposti kohdealustan mukaan.

Hooks-kansion alapuolella kuviossa 5 on node_modules-kansio, joka sisältää kaikki npm-pakettienhallinnan avulla haetut ohjelmistoriippuvuudet. Npm on Node.js pohjaisten JavaScript-sovellusten pakettienhallinta, joka on samalla maailman suurin ekosysteemi avoimen lähdekoodin kirjastoille. Node.js on taas JavaScriptiä suorittava ohjelmaympäristö, joka on rakennettu Googlen V8 JavaScript-moottorin päälle. Näiden avulla Ionic ja monet muut kehykset ja kirjastot ylläpitävät riippuvuuksia muihin kirjastoihin ja ohjelmakoodeihin. Node_modules-kansiosta löytyykin sovelluskehyksen asentamisen jälkeen esimerkiksi Ionic Frameworkin ohjelmakoodi, sekä Angularin ohjelmakoodi ja sen vaatimat JavaScript-kirjastot. (19.)

Kansiorakenteen juuressa oleva package.json-tiedosto määrittelee Ionic Frameworkin ensimmäisen tason riippuvuudet, jokainen näistä voi kuitenkin sisältää samaan tapaan riippuvuuksia muihin kirjastoihin. Npm lukee nämä tiedot ja lataa vaadittavat paketit automaattisesti node_modules-kansioon asennuksen yhteydessä. Muut tiedostot, esimerkiksi gulpfile.js, webpack.config.js ja config.xml määrittelevät muun muassa sovelluksen pakkaukseen ja kokoamiseen liittyviä asioita.

3.4.2 Resurssit

Projektin alustakohtaiset resurssitiedot, eli käytännössä sovelluskuvakkeet ja splash-latauskuva, löytyvät resources-kansion alta. Samaten sovelluksen eri alustoille sitovat natiivit ohjelmakoodit sekä koko käännetty sovellus löytyy platforms-kansiosta jaoteltuna

alakansioihin tuettavien alustojen mukaan. Kuviossa 5 oikealla näkyykin käännetyn sovelluksen Android- ja iOS-versioiden omat projektirakenteet platforms-kansion alla.

Sovellus käännetään kohdelaitteille vain erikseen käskettäessä, koska operaatio vie jonkin verran aikaa. Sen sijaan muutosten nopeaan tarkasteluun ja lennosta muokkaamiseen sovelluksen verkkosovellusosa käännetään ensin www-kansioon, josta löytyy käytännössä puhdas verkkosivuversio sovelluksesta. Sen avulla voidaan testata muutoksia nopeasti selaimessa tai kohdelaitteessa. Kuitenkin natiivit rajapinnat saadaan kytkettyä sovellukseen vasta käännettäessä se erikseen kohdelaitealustalle. (20.)

3.5 Ionic framework 2 & Angular 2

Ionic-sovelluskehys rakentuu Angular-sovelluskehyksen päälle jolloin Angularin muutokset vaikuttavat suoraan myös Ionic Frameworkiin. Niinpä kun Angularista julkaistiin alfatasoinen 2. versio, myös Ionic-kehyksestä lähdettiin rakentamaan toista versiota, joka on kirjoitushetkellä edennyt pitkälle beta-vaiheeseen. Kuitenkin Ionic Frameworkin kehittäjien omien sanojen mukaan kehyksen 2. versio on jo nyt valmis tuotantokelpoisten sovellusten kehittämiseen (21).

Angular 2 tuo mukanaan paljon muutoksia kehyksen ensimmäisiin versioihin. Kehys käyttää ECMAScriptin versiota 6, tällä hetkellä selaimien tukeman 5. version sijaan. ECMAScript on standardi, jota JavaScript ja sitä suorittavat selainmoottorit käyttävät (19). Kuudes versio tuo mukanaan useita uudistuksia ja toimiakseen nykyisillä selaimilla, se tulee ensin kääntää standardin viidennen version mukaiseksi. Kääntäminen on kuitenkin automatisoitavissa, joten uutta standardia päästään hyödyntämään Angular 2 -pohjaisissa projekteissa suoraan.

Angular 2 on kirjoitettu käyttäen Microsoftin luomaa TypeScript-ohjelmointikieltä, mikä on muunnelma perinteisestä JavaScriptistä (19). TypeScript käyttää tietorakenteiden tyyppien määritykseen staattista tyypitystä eli sovellusta kirjoitettaessa esimerkiksi olioiden tyyppi tulee esittää ohjelmakoodissa JavaScriptin dynaamisen tyypityksen sijaan. TypeScriptin mukana tulee myös kääntäjä, jolla sillä kirjoitetut sovellukset voidaan kääntää selainten tukemaan ECMAScript 5 -standardiin. Sovelluksia voi kuitenkin kirjoittaa perinteiselläkin JavaScriptillä pienillä muutoksilla, mikä voi aiheuttaa myös ongelmia, kuten luvussa 4.2 ilmenee.

4 Kehitysprosessi Ionic Frameworkilla

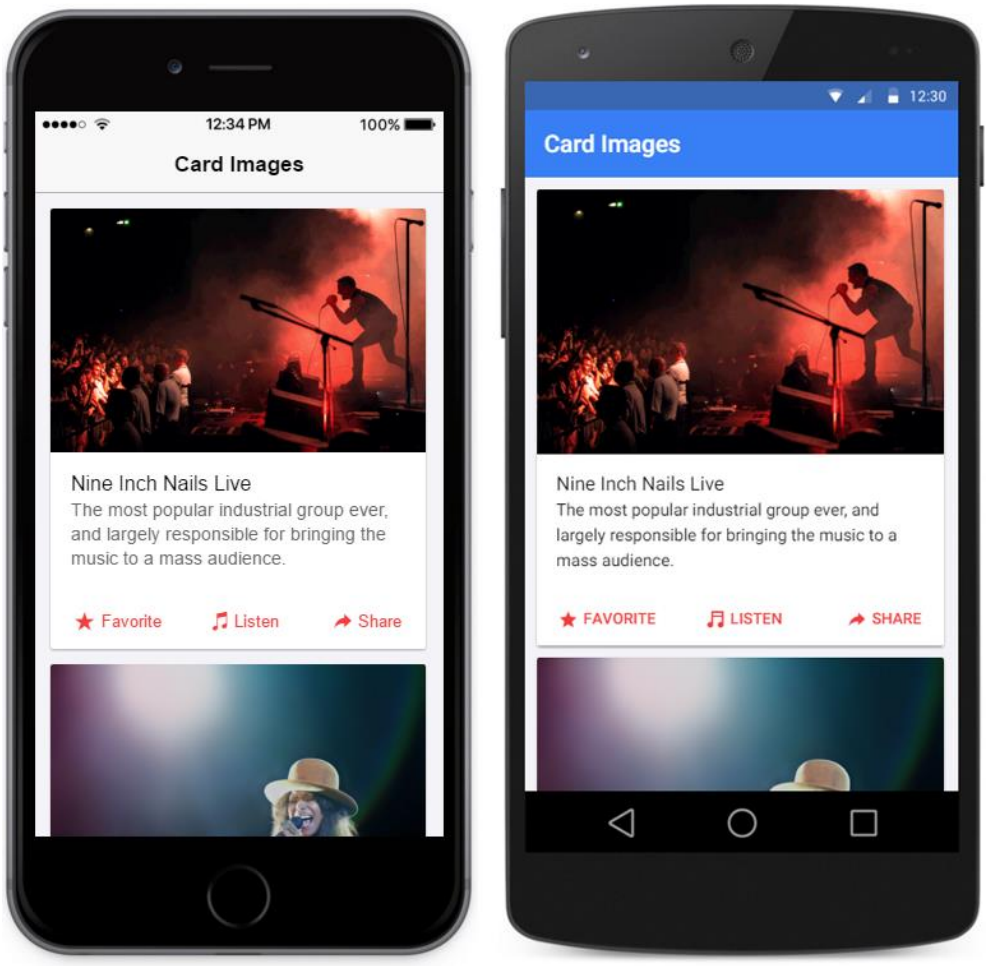
Ionic asennetaan npm-moduulina, jolloin ainoaksi lähtövaatimukseksi muodostuu Node.js:n asentaminen. Ajettaessa npm:n asennuskomento, Ionic lataa automaattisesti tarvittavat riippuvuudet ja lähdekoodit käyttöä varten. Tämän jälkeen Ionic Frameworkin verkkosivuilla suositellaan valmiin aloitusprojektin asentamista. Aloitusprojekteja on tarjolla useita, mutta beta-vaiheessa olevaan 2. versioon kirjoittamishetkellä vain yksi. Projektin asentaminen käy helposti komentorivikomennolla ”ionic start <projektinNimi>”. Kaikki muut komentorivipohjaiset työkalut on esitelty tarkemmin luvussa 4.3. (22.)

4.1 Käyttöliittymäkomponentit

Ionic Framework tarjoaa sovellusrungon ja perustoiminnallisuuksien lisäksi joukon käyttöliittymäkomponentteja, joilla voi nopeasti rakentaa käyttöliittymiä mobiilisovelluksille. Jokaisella komponentilla on oma ulkoasunsa sekä iOS- että Android-versiolle, mutta niiden toiminnallisuus on yhtenevä. Esimerkiksi hakupalkki toimii samoin sekä Androidilla että iOS:lla, mutta näyttää hieman erilaiselta. Komponentit ovat myös tarvittaessa tyyliteltävissä kokonaan uudelleen. Tässä luvussa esitellään muutamia yleisimpiä käyttöliittymäkomponentteja ja niihin liittyviä käyttöliittymän suunnittelumalleja. Samoja suunnittelumalleja ja käyttöliittymäkomponentteja käytetään myös Intellian sovellusprojektissa.

Kortit

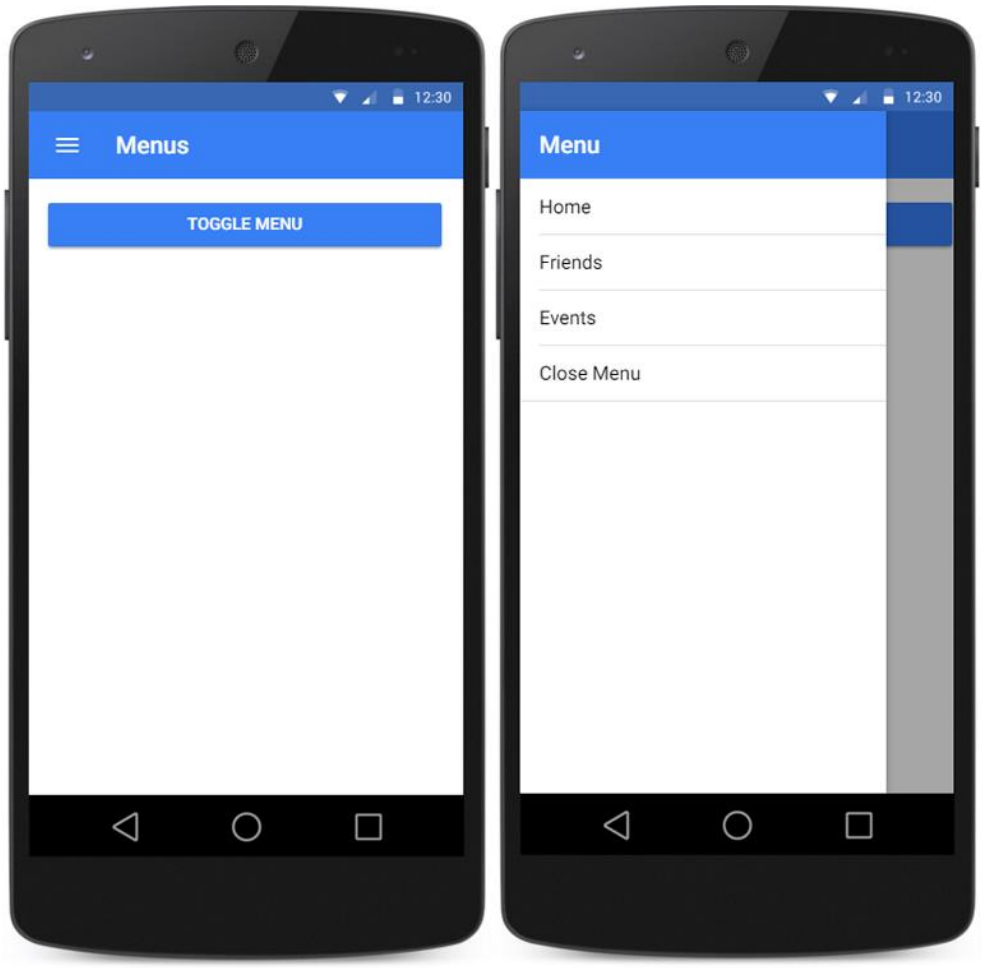
Kortit ovat nopeasti yleistynyt tapa esittää tietosisällön eri osioita, erityisesti pienellä ruudulla. Kortit mahdollistavatkin helpohkon tavan jäsennellä sama sisältö erikokoisille näytöille, sillä niitä voi helposti asetella allekkain tai vierekkäin ja suurentaa tai pienentää tarpeen mukaan. Korttien sisään voi asettaa myös käyttäjän interaktiota vaativia osioita, kuten karttoja ja sosiaalisten medioiden liitännäisiä. Kuviossa 6 on esimerkki kuvallisesta kortista Ionic Frameworkissa sekä iOS- että Android-käyttöjärjestelmillä. (23.)



Kuvio 6. Kortit-käyttöliittymäkomponentti Ionic Frameworkissa (23)

Valikot

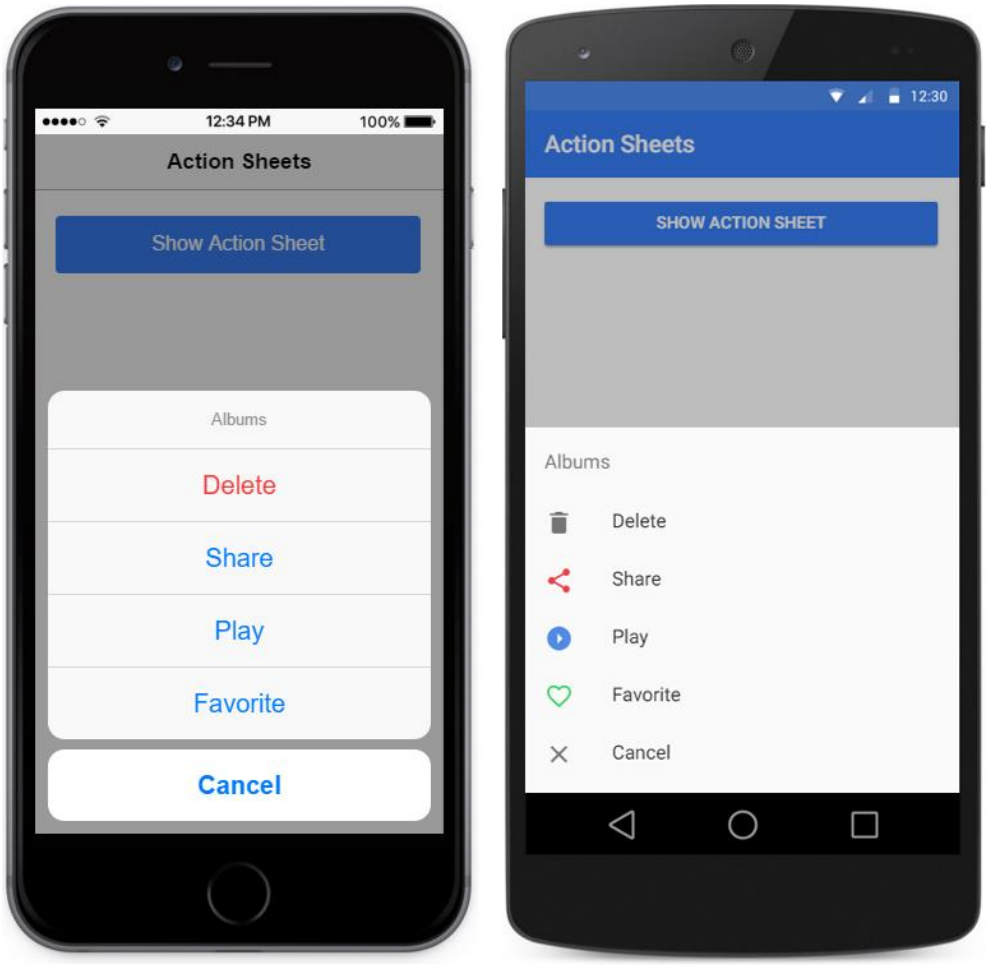
Sivuvalikko on yksinkertainen tapa järjestää mobiilisovelluksen navigointi. Sillä saadaan pääsisällölle tilaa näkymässä ja samalla lajiteltua sovelluksen informaatiosisältö järkevämmin kuin näyttämällä kaikki navigointimahdollisuudet kerralla käyttäjälle. Lisäksi käyttöliittymäkomponentin ollessa yleisesti käytössä niin verkkosivuilla kuin sovelluksissakin, saadaan sitä käyttämällä aikaiseksi parempi käyttökokemus sovellukselle komponentin tuttuuden myötä. Ionic huomioi myös eri alustojen natiivit erot valikoiden toiminnassa, sillä ne aukeavat hieman eri tavoilla eri käyttöjärjestelmillä. Kuviossa 7 esitellään valikon toiminta Android-käyttöjärjestelmällä (23.)



Kuvio 7. Vasemman sivupalkin toiminta Ionic Frameworkissa Android-käyttöjärjestelmällä (23)

Toimintovalikko

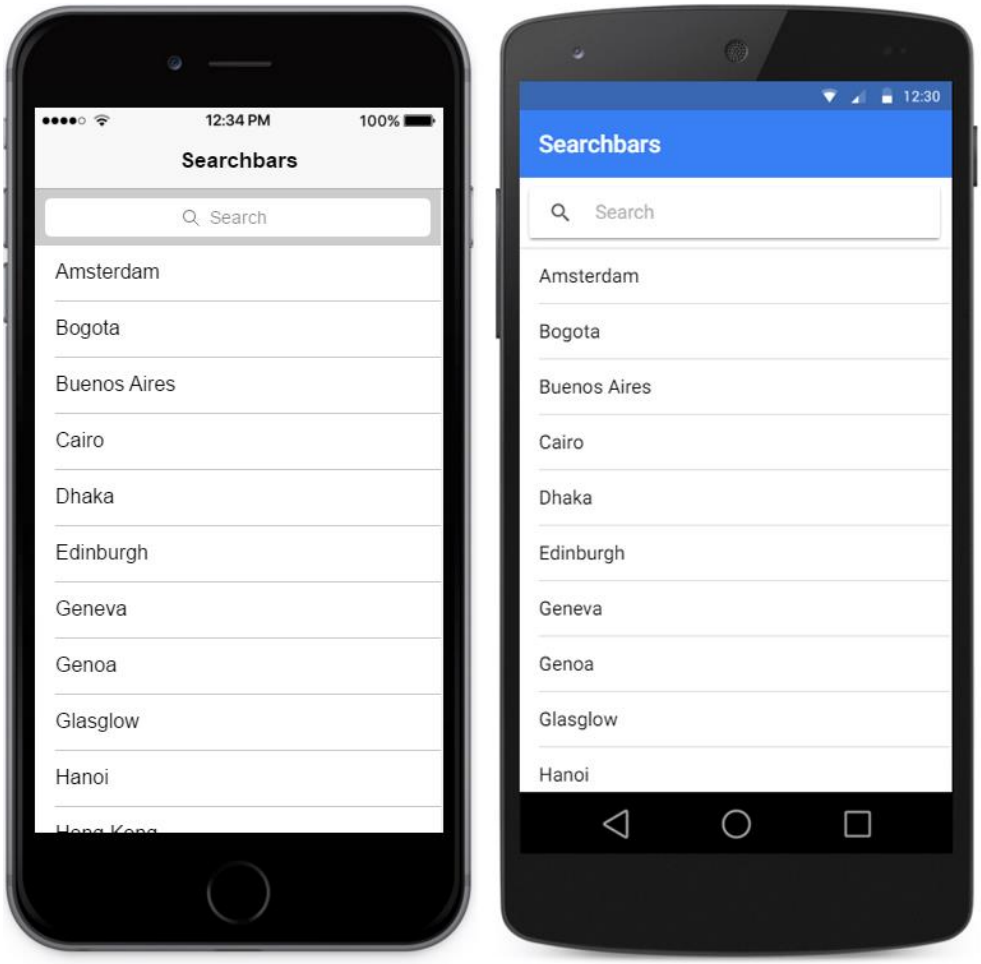
Toimintovalikolla tarkoitetaan erilaisten toimintojen niputtamista omaan valikkonäkymäänsä. Esimerkiksi yhteystietoihin liittyvä toiminto voi avata valikon, jonka vaihtoehtoina on yhteystiedon lisääminen puhelimen muistiin tai soittaminen yhteystiedolle. Kuviossa 8 on toimintovalikon oletusnäkymä sekä iOS- että Android-käyttöjärjestelmillä. (23.)



Kuvio 8. Toimintovalikko Ionic Frameworkissa (23)

Hakupalkki

Hakutoiminnallisuus on yksi yleisimmistä toiminnoista mobiilisovelluksissa. Sovellusten hakutoiminnallisuus on kuitenkin usein integroitu sovellusten näkymän yläpalkkiin tai erilliselle sivulle. Ionic Frameworkin käyttämä hakupalkki noudattaa alustojensa natiiveja tyylejä ja voi lisäksi automaattisesti avata päätelaitteen näppäimistön käyttäjän näpäyttäessä hakupalkkia. Kuviossa 9 on hakupalkin iOS- ja Android-versio. (23.)



Kuvio 9. Hakupalkki Ionic Frameworkissa (23)

4.2 Komponentit, dekoraattorit ja näkymät

Ionic Frameworkin 2. version ollessa toteutettuna Angular 2:n ympärille, sillä luodut sovellukset rakennetaan modulaarisesti komponenteista. Esimerkiksi yksi sivu tai sivun sisällä oleva näkymä voi sisältää erillisiä alakomponentteja omine näkymineen. Komponentit ovat yksi ilmentymä direktiiveistä Angular 2:ssa. Direktiivit ovat Angularin keskeisin ominaisuus. Niiden avulla sidotaan selaimessa suoritettava HTML-koodi määritettyjen direktiiviluokkien instansseihin. Direktiivi on yleensä sidottu itse nimettyyn HTML-elementtiin tai HTML-attribuuttiin, jolloin kehittäjän omat direktiivit ikään kuin laajentavat tavanomaista HTML:ää. Näkymään sitomisen lisäksi direktiiveillä on omat esitellyt riippuvuutensa JavaScript-moduuleihin, oma konstruktorinsa, metodinsa sekä tarvittaessa erilaisia dekoraattoreiden avulla määriteltyjä ominaisuuksia (24).

Dekoraattori on suunnittelumalli, joka on otettu vahvasti mukaan Angular 2:ssa, jolloin se on hyödynnettävissä myös Ionic Frameworkissa. Gamman, Helmin, Johnsonin ja Vlissides'n Design Patterns -kirjassa määritellään dekoraattori yksinkertaisesti siten, että se mahdollistaa uusien toiminnallisuuksien lisäämisen luokan instanssiin muuttamatta sen alkuperäistä rakennetta (25, s. 175). Dekoraattoreiden avulla voidaankin yksinkertaisesti esimerkiksi erottaa kokonaisen sivun sisältävä komponentti pelkästä yhden toiminnon sisältävästä direktiivistä tai esimerkiksi palveluluokasta (26). Alla olevassa esimerkkikomponentissa esitellään lista-komponentin ohjelmakoodi. Komponentti on mukaelma Ionic 2:n aloitusprojektista. (27.)

```
import {IonicApp, Page, NavController} from 'ionic-angular';

@Page({
  templateUrl: './pages/list/list.html'
})
export class ListPage {
  static get parameters() {
    return [[IonicApp], [NavController]];
  }
  constructor(app, nav) {
    this.app = app;
    this.nav = nav;
    this.items = [];

    for(let i = 1; i < 11; i++) {
      this.items.push({
        title: 'Item ' + i,
        note: 'This is item #' + i
      });
    }
  }

  itemTapped(event, item) {
      this.nav.push(ItemDetailsPage, {
        item: item
      });
  }
}

```

Esimerkkikoodi 1. Lista-komponentin sovelluslogiikka Ionic Frameworkissa (27)

Esimerkkikoodissa 1 esitellään aluksi riippuvuudet muihin luokkiin import-komennolla. Tämän lisäksi riippuvuudet otetaan käyttöön itse komponentissa injektoimalla ne komponentin konstruktoriin parameters-funktion avulla rivillä 7 ja 10. Listakomponentilla on nyt pääsy muihin komponentteihin, ja ne voidaan asettaa luokkatason muuttujien arvoiksi myöhempää hyödyntämistä varten.

Riippuvuudet voitaisiin määritellä myös käyttämällä Angular 2:n @Inject-dekoraattoria, mutta Ionic Framework käyttää tähän tarkoitukseen parameters-funktiota. Tämä johtuu siitä, että funktion parametrina annettavat dekoraattorit eivät kääntämisen ongelmien vuoksi toimi kunnolla, mikäli sovelluksia kirjoitetaan käyttäen perinteistä JavaScriptiä (28). Dekoraattorin avaaminen funktioksi havainnollistaa kuitenkin hyvin dekoraattorin rakenteen JavaScript-koodissa. Dekoraattorit ovat käytännössä funktiota, joita voidaan kutsua @-merkillä alkavien annotaatioiden avulla ja niillä on mahdollista muokata luokan ilmentymän toiminnallisuutta suoraan. Inject-dekoraattorin tapauksessa luotavaan olioon voidaan lisätä muiden luokkien instansseja käytettäväksi.

Luokkamääreen yläpuolella olevassa Page-dekoraattorissa määritellään kyseisen komponentin toimivan sivuna sovelluksessa. Lisäksi dekoraattorin sisällä määritellään komponentille näkymän URI-osoite. URI eli Uniform Resource Identifier on merkkijono, jolla esitetään tietyn resurssin käyttämä protokolla ja sisällön osoite. Protokolla on kuitenkin vapaaehtoinen tieto, ja tässä tapauksessa sitä ei tarvita. URI:n tunnetuin muoto on verkkodokumenttien sijainnin kertomiseen käytettävä Uniform Resource Locator (URL), joka eroaa URI:sta siten, että siinä on aina mukana käytettävä protokolla.

URI:n avulla viitattu näkymä on itsessään HTML-koodia, mutta Angular tuo siihen kaksisuuntaisen tiedonsitomisen lisäominaisuutena. Tämä tarkoittaa sitä, että kun komponentin sisällä olevan muuttujan arvo vaihtuu, Angular piirtää automaattisesti muuttuneen arvon näkymään sille osoitetulle paikalle. Sama toimii myös toiseen suuntaan, eli arvon vaihtuessa esimerkiksi näkymän syöttökentässä se vaihtuu automaattisesti myös komponentista luodussa JavaScript-olion instanssissa.

Pelkkien arvojen muuttamisen lisäksi Angular tarjoaa muun muassa erilaisia toistorakenteita ja ehtolauseita tiedon näyttämisen tueksi, sekä tuen käyttöliittymän erilaisia tapahtumille, kuten elementtien klikkaamiselle, näppäinten painalluksille ja hiiren liikkeille. Tapahtumien kuuntelu ja tiedon näyttäminen rakennetaan komponenttien näkymiin. Esimerkkikoodissa 2 on esitelty esimerkkikoodin 1 listakomponentin näkymä.

```
1   <ion-navbar *navbar>
2       <button menuToggle>
3           <ion-icon name="menu"></ion-icon>
4       </button>
5       <ion-title>List</ion-title>
6   </ion-navbar>
7
8   <ion-content>
9       <ion-list>
10          <button ion-item
11                  *ngFor="#item of items"
12                  (click)="itemTapped($event, item)">
13              {{item.title}}
14              <div class="item-note" item-right>
15                  {{item.note}}
16              </div>
17          </button>
18      </ion-list>
19  </ion-content>
20
```

Esimerkkikoodi 2. Koodin 1 listakomponentin näkymän ohjelmakoodi (27)

Näkymän sisällä muuttujien arvot sidotaan kaksilla kaarisuluilla HTML-koodiin halutuille paikoille. Listakomponentin tapauksessa listan alkion otsikko ja viesti sidotaan riveillä 13 ja 15. Lisäksi käytetään toistorakennetta *ngFor listan iteroimiseksi. Näkymässä on käytetty myös paljon kehyksen tarjoamia direktiivejä näkymän rakentamiseen. Ne ovat tunnistettavissa ion-alkuisista elementeistä ja attribuuteista.

Näkymässä jokaiselle listan alkiolle lisätään tapahtumankuuntelija click-tapahtumalle toistorakenteessa rivillä 11. Tapahtumien kuuntelija välitetään automaattisesti koodin 1 samannimiseen metodiin, jolloin alkiota klikattaessa voidaan suorittaa metodin sisällä oleva ohjelmakoodi. Listakomponentissa klikattaessa listan alkiota, komponentti lisää navigaatioon uuden sivun, mihin välitetään klikattu alkio. Näin klikattu rivi avaa uuden sivun sovelluksessa nykyisen sivun päälle, jossa voidaan esimerkiksi näyttää alkion tarkemmat tiedot.

4.3 Palvelut ja Observer-suunnittelumalli

Ionic Framework tukee sivujen lisäksi myös palvelumuotoisia komponentteja, joiden avulla esimerkiksi rajapintakutsujen rakentaminen helpottuu. Palvelut ovat itsessään

Singleton-tyyppisiä suunnittelumalliltaan, joten niitä voi olla olemassa vain yksi instanssi kerrallaan (25, s. 127). Palvelut injektoidaan sovelluksen käyttöön samaan tapaan kuin muutkin komponentit, mutta ne pitää lisäksi määritellä App-direktiivin sisään providers-taulukkoon, jotta sovelluskehys osaa käyttää niitä oikein (24). Esimerkkikoodissa 3 esitellään yksinkertaistettu esimerkkipalvelu kirjautumista varten.

```
1  import {Injectable} from 'angular2/core';
2  import {Http} from 'angular2/http';
3  import 'rxjs/add/operator/map';
4
5  @Injectable()
6  export class LoginService {
7      static get parameters() {
8          return [[Http]];
9      }
10
11     constructor(http) {
12         this.http = http;
13         this.endpoint = "http://example.com/login";
14         this.loggedIn = false;
15
16         this.authenticate();
17     }
18
19     authenticate() {
20         this.http.get(this.endpoint, {headers: {'Accept': 'application/json'}})
21             .map(res => res.json())
22             .subscribe(
23                 result => this.loggedIn = result.LoginResult,
24                 error => console.log(error));
25     }
26 }
27
28
```

Esimerkkikoodi 3. Esimerkki palvelusta Ionic Frameworkissa

Palvelussa määritellään aluksi sen riippuvuudet import-riveillä, jonka jälkeen rivillä viisi määritellään palvelun olevan injektoitavissa muihin luokkiin Injectable-dekoraattorin avulla. Dekoraattori toimii yhdessä luokan export-määreen kanssa siten, että palvelusta muodostetaan ainoastaan yksi instanssi sovelluksen suorituksen aikana. Esimerkin luokalla on yksi metodi, authenticate, joka simuloi autentikoitumista http-kutsun avulla. Rivillä 20 metodi käyttää http-kutsuun Angular 2:n mukana tulevaa Rx.js-kirjastoa, jossa käytäntönä on mukautetun Observer-suunnittelumallin käyttö asynkronisissa kutsuissa (29).

Observer-suunnittelumallilla tarkoitetaan rakennetta, jossa yhden olion muuttaessa tilaansa siitä lähetetään tieto muille sitä kuunteleville olioille, jotta ne voivat muuttaa tilaansa vastaavasti. Tällä tavalla on mahdollista sitoa esimerkiksi eri komponentteja sovelluksen tilan muutoksiin ilman tarvetta rakentaa erillistä periytyvyyttä. Suunnittelumalli määrittelee kaksi eri rakennetta mallin toiminnan mahdollistamiseksi: kohteen (Subject) sekä sen tarkkailijan (Observer). (25, s. 293.)

Palveluesimerkissä Observer-malli toimii siten, että http-metodin kutsu palauttaa Observable-luokan instanssin, jonka julkaisemiin tapahtumiin kiinnitytään sen subscribe-metodilla. LoginService-luokka toimii tarkkailijana ja itse http-kutsu on tarkkailun kohde. Rivillä 20 aloitetaan http-kutsun suorittaminen määritellyillä header-parametreillla, jonka jälkeen kutsutaan tulokselle ensin Rx.js:n map()-metodia. Map-metodissa muunnetaan kutsun tuloksen tietotyyppi automaattisesti JavaScript Object Notation (JSON) -muotoisesta tekstistä JavaScript-olioksi, jotta tuloksia voidaan käsitellä suoraan ohjelmakoodissa. Tämän jälkeen kutsun tapahtumiin liitytään subscribe-metodilla.

Riveillä 23 ja 24 käsitellään kutsun onnistuminen ja epäonnistuminen omissa funktioissaan. Esimerkissä onnistunut kirjautumispyyntö muuttaa loggedIn-totuusarvon todeksi ja epäonnistunut pyyntö välittää epäonnistumisen syyn selaimen konsoliin. Rx.js mahdollistaa paljon muitakin asynkronisiin kutsuihin liittyviä toimintoja. Niitä esitellään enemmän luvussa 5.2.1.

4.4 Kehitystyökalut

Ionic on kehittänyt itse sovelluskehyksen lisäksi erilaisia hyödyllisiä komentorivipohjaisia työkaluja, joilla kehitystyön eri vaiheita saadaan nopeutettua ja automatisoitua. Taulukossa 1 esitellään työkalun käynnistävä komentorivipohjainen komento sekä mitä työkalulla voidaan tehdä.

Taulukko 1. Ionic:n komentorivityökalujen komennot ja niiden selitteet (20)

ionic platform add <alusta>	Kiinnittää Ionic:n alustamoduulin Cordovaan. Toimenpide tulee suorittaa ennen ”ionic run” -komentoa, mutta Ionic suorittaa komennon automaattisesti tarvittaessa itsekin. Kohdan <alusta> paikalle voi valita minkä tahansa Cordovan tukeman alustan.
ionic run <alusta>	Kääntää, asentaa ja suorittaa sovelluksen kohdealustalle. Mikäli tietokoneeseen ei ole kytketty alustan mukaista laitetta, sovellus yritetään suorittaa alustan emulaattorissa.
ionic emulate <alusta>	Kääntää, asentaa ja emuloi sovelluksen kohdealustalle.
ionic serve	Rakentaa verkkosivuversion sovelluksesta, joka käynnistetään omalle paikalliselle palvelimelle. Samalla lähdekoodin muutoksien tunnistaminen otetaan käyttöön, jolloin koodin muuttuessa uusi versio rakennetaan ja sitä suorittava selainikkuna päivitetään automaattisesti.
ionic resources	Luo automaattisesti sovelluskuvakkeen ja latausruudun kaikille lisätyille alustoille eri näyttöresoluutiot huomioiden lähdekuvatiedostosta.

Komentorivipohjaisten työkalujen lisäksi kehitystyötä nopeuttaa ja parantaa JavaScript-ohjelmistokehitykseen tarkoitetun IDE:n käyttö. IDE eli Integrated Development Environment on sovelluskehitysympäristö, jossa vähintään koodin kääntäjä on sovitettu yhteen koodieditorin kanssa. Tässä opinnäytetyössä sovelluskehitykseen on käytetty JetBrains Webstorm -editoria, minkä käytöstä kaikki työssä esitellyt koodiesimerkit ovat kuvankaappauksia.

Apuna Intellian mobiilisovelluksen rakentamiseen ja koodin suorituksenaikaiseen tarkasteluun käytettiin Google Chrome -selainta, jonka mobiilinäkymän avulla sovelluksesta on saatu otettua kaikki kuvankaappaukset. Chromen kehitystyökalut on lisäksi mahdollista kytkeä toimimaan Android-puhelimen kanssa suoraan, jolloin suoritettavaa koodia voi tarkastella samaan aikaan, kun sovellusta suoritetaan mobiililaitteessa.

5 Intellia yritys- ja yhteystietohakusovellus

Intellia Oy on vuonna 2000 perustettu erilaisten asiakasrekisterien keräämiseen ja ylläpitoon erikoistunut yritys (30). Yrityksen liikevaihto oli vuoden 2014 lopussa 1,8 miljoonaa euroa, ja yritys työllistää kahdeksan ihmistä (31). Intellia tarjoaa esimerkiksi räätälöityjä kohderyhmiä, liidejä sekä erilaisia työkaluja yritysten kontaktointiin. Tämän lisäksi Intellia tarjoaa rekisterien päivityksiä, sähköpostituksia sekä asiakasanalyyseja. (32.)

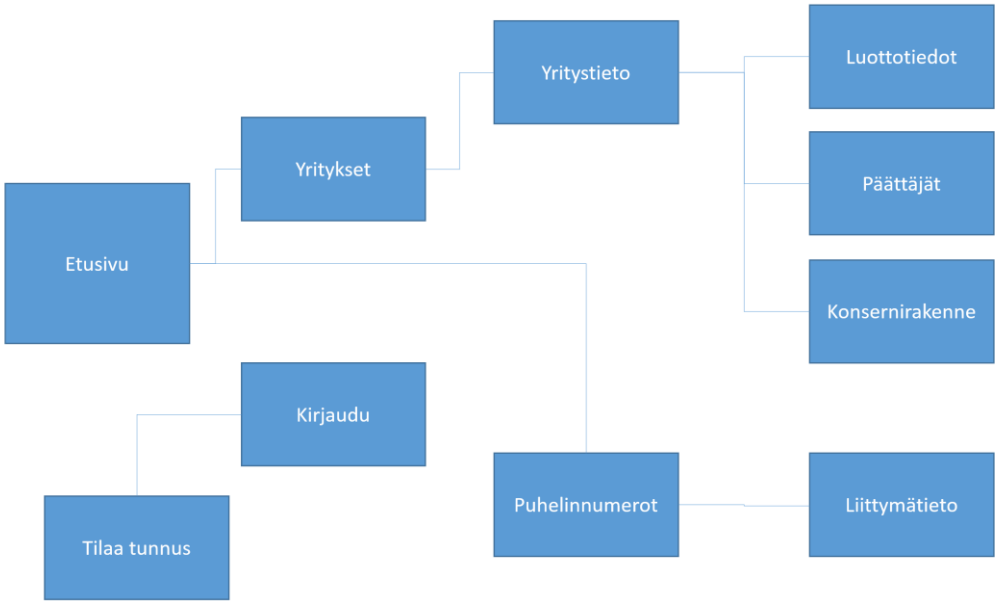
Intellialla on jo olemassa kattavat asiakastietopalvelut verkossa, mutta ne ovat huonosti käytettävissä mobiililaitteilla palveluiden ollessa suunniteltu erityisesti työpöytäkäyttöön. Lisäksi palveluiden työkalut ovat toiminnoiltaan sellaisia, että niille on vaikea löytää käyttötapauksia mobiililaitteille. Mobiilisovelluksen tavoitteena onkin tuoda Intellian palveluita myös mobiililaitteille soveltuvin osin.

Toteutettava mobiilisovellus rakentuu lähtökohdiltaan Intellian yritys- ja yhteystietorekistereiden ympärille. Näitä voidaan käyttää mobiililaitteilla hakupalveluna, jonka hakutuloksia voi suoraan liittää mobiililaitteiden perustoiminnallisuuksiin. Haun avulla löydetylle yritykselle voi esimerkiksi soittaa, laittaa sähköpostia tai vierailla sen verkkosivuilla. Samanlaisia toiminnallisuuksia voidaan rakentaa myös yhteystietoja varten. Sovelluksen kantavana ajatuksena onkin luoda helposti käytettävä hakupalvelu, jolla käyttäjä löytää nopeasti etsimänsä tiedon ja voi hyödyntää sitä mutkattomasti.

Lähtökohtana sovellukselle on valmis Representational State Transfer (REST) -suunnittelumallia noudattava hakurajapinta sekä alustava tiedon jäsentelysuunnitelma. Tässä luvussa käsitellään sovelluksen suunnittelua käyttöliittymän ja toiminnallisuuksien osalta ja käydään läpi sovelluksen kehitystä vaihe vaiheelta. Erityisesti keskitytään haku- ja hybridimobiilisovelluksille tyypillisiin haasteisiin, kuten automaattisen haun toteuttamiseen ja alustakohtaisiin eroihin mobiililaitteen omien toimintojen liittämisessä sovellukseen. Lopuksi käydään läpi sovelluksen käytettävyyden ja käyttökokemuksen parantamista ja tarkastellaan syntynyttä sovellusta kokonaisuutena.

5.1 Sovelluksen suunnittelu

Intellian mobiilisovelluksen suunnittelun aloittamisen lähtökohtana on aiemmin hahmoteltu sivukartta sovelluksen eri näkymistä, mikä esitellään kuviossa 10. Sovelluksen pääsivuna toimii hakusivu, jolta on mahdollista hakea sekä yrityksiä ja yhteystietoja. Yritys-hakutuloksille sovellus tarjoaa perustietojen lisäksi tietoa yrityksen päättäjistä ja konsernirakenteesta sekä yrityksen taloustietoja, mikäli ne ovat saatavilla. Yhteystiedoille on saatavilla joitakin liittymätietoja. Näiden lisäksi sovelluksella on oma kirjautumissivu sekä sivu tunnuksen tilaamiselle.



Kuvio 10. Mobiilisovelluksen sivukartta suunnitteluvaiheessa

Sovelluksen tarjoamien tietojen organisoimisen lisäksi suunnitteluvaiheessa huomioitiin kohdelaitteiden tarjoamat mahdollisuudet suunnittelemalla niihin integroitavia toimintoja, kuten yhteystietojen lisäämisen laitteen osoitekirjaan ja karttasovelluksen käynnistämisen yrityksen osoitteeseen. Sovellukseen voisi rakentaa myös esimerkiksi soittajan tunnistamistoiminnon tai automaattisen yhteystietojen päivittämisen laitteen osoitekirjaan, mutta ne rajattiin pois jatkokehitysmahdollisuuksiksi.

Sovelluksen rekisteröitymistoimintoa pohdittiin monelta eri kantilta ja päädyttiin toteuttamaan yksinkertainen sähköpostipohjainen rekisteröityminen. Kuitenkin sovellukselle halutaan tarjota rajoitettu julkinen käyttöoikeus sen testaamisen mahdollistamiseksi.

Käyttöliittymän ja ulkoasun suunnittelun perusta muodostui esimerkkitarkoitukseen tehdystä käyttöliittymädemonstraatiosta. Tästä versiosta luovuttiin kuitenkin, ja hakutoiminnallisuus yhtenäistettiin yksinkertaisempaan kokonaisuuteen. Käyttöliittymän kantavana teemana haluttiin pitää toimintojen suorittamisen helppous ja mutkattomuus, sekä selkeä ja helppolukuinen ulkoasu. Kuviossa 11 esitellään muutos alkuperäisestä ulkoasusta sovelluksen viimeisimpään ilmeeseen vasemmalta oikealle.

Kuvio 11. Sovelluksen ulkoasun ensimmäinen versio (vasemmalla) ja viimeisin versio (oikealla)

5.2 Sovelluksen kehittäminen

5.2.1 Hakukomponentti

Intellian mobiilisovelluksen tärkein toiminto on haku, joten sen tulee toimia nopeasti, suoraviivaisesti ja luotettavasti. Samalla hakupalkki on myös ensimmäinen komponentti, jota käyttäjä sovelluksessa käyttää, jolloin sillä on tärkeä rooli ensivaikutelman luomisessa. Hakukomponentti muuttuikin jo varhaisessa vaiheessa kehitystä kahdesta erillisestä hakukentästä yrityksille ja yhteystiedoille yhtenäiseksi ratkaisuksi, jossa käyttäjä voi hakusanan kirjoittamisen jälkeen päättää ja vaihtaa tarvittaessa haun kohdetta kirjoittamatta hakusanaa uudelleen. Toiminto on esitelty kuviossa 12.

Kuvio 12. Yhdistetty hakutoiminnallisuus

Hakujen yhdistämisen lisäksi hakupalkista piilotettiin erillinen hakupainike kokonaan. Erillisen nappulan painaminen vie käyttäjältä turhaan aikaa, sillä toiminto voidaan suorittaa automaattisesti ennalta määrätyn ajan kuluttua hakutermin kirjoittamisen lopettamisesta ja lisäksi painamalla näppäimistön Enter-nappia. Samalla haku muuttuu interaktiivisemmaksi, sillä käyttäjä näkee hakutuloksia heti, kun kirjoittamiseen tulee tauko. Tämä auttaa käyttäjää rajaamaan hakua, kun haun tulokset ovat heti saatavilla.

Näiden vaatimusten toteuttamiseen luvussa 4.3 esitelty Rx.js-kirjasto tarjoaa pitkälle vietyjä ratkaisuja. Esimerkkikoodissa 4 esitellään hakulogiikan toteutus yksinkertaistettuna hakukomponentin sisällä.

Esimerkkikoodi 4. Hakulaatikko ja sen yksinkertaistettu toteutus

Esimerkkikoodin 4 yläosassa olevassa HTML-koodissa käytetään Ionic-kehyksen omaa ion-searchbar-direktiiviä hakulaatikon piirtämiseen. Kutsumalla direktiiviä HTML-sivulla Angular osaa hakea siihen määritellyt ulkoasu- ja sovelluslogiikat automaattisesti: tässä tapauksessa luoda tekstikentän ja sitoa haun tyhjentämisen (clear) ja peruuttamisen (cancel) liittyvät metodit. Direktiiviin sidotaan lisäksi hakutermille oma malli [ngFormControl]-ilmaisulla. Nyt hakusanan muutokset päivittävät automaattisesti JavaScript-koodissa määriteltyä ”term”-muuttujaa.

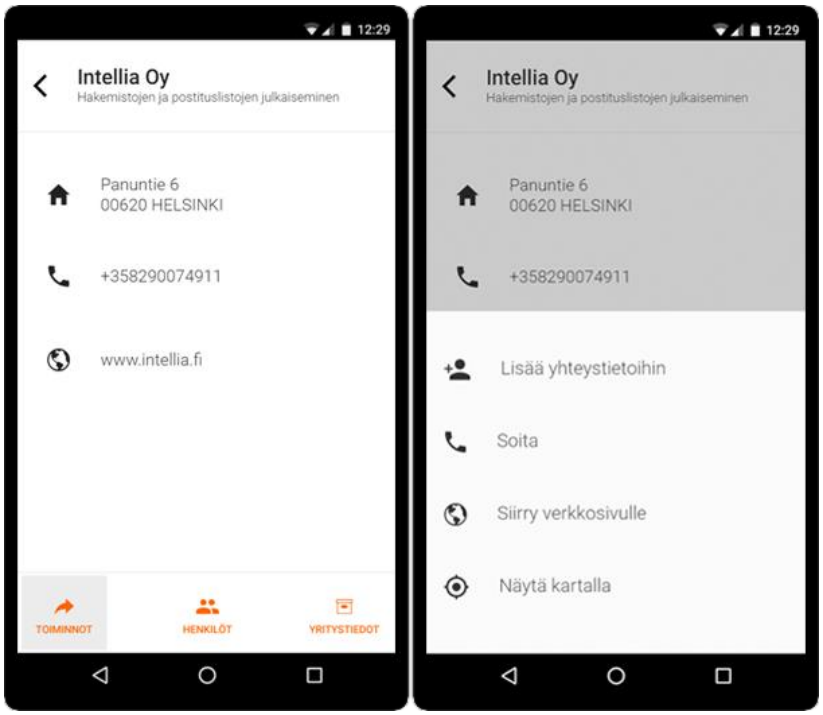
JavaScript-koodissa käsitellään hakusanan muutoksia Rx.js:n ketjutusten avulla. Rivillä 2 määritellään, että ketjua jatketaan eteenpäin aina, kun hakusana muuttuu. Rivillä 3 suodatetaan pois muutokset, joissa syötteen pituus on pienempi kuin kaksi merkkiä.

Mikäli hakutermin muutos läpäisee aiemman suodattimen, kutsutaan seuraavaksi rivillä 4 debounceTime-metodia erillisellä numeroarvolla. Debounce toimii yksinkertaistettuna

siten, että se sallii arvon lähetyksen eteenpäin ketjussa vasta, kun se on säilynyt muuttumattomana ennalta määrätyn ajan verran. Toteutuksessa aika luetaan konfiguraatiotiedostosta ja on tässä tapauksessa 600 millisekuntia. Käyttämällä debounceTime-metodia jokainen merkin painallus ei tee turhaan uutta hakua ja haun suorittaminen voidaan silti automatisoida. Mikäli hakusanan muutos läpäisee ketjun kaikki vaiheet, kutsutaan varsinaista hakumetodia, joka muodostaa yhteyden hakurajapintaan ja käsittelee hakutulokset. (29.)

5.2.2 Integraatio laitealustan toimintoihin

Onnistuneiden hakujen toteuttamisen lisäksi sovelluksen tulee voida hyödyntää laitealustan toimintoja suurimman mahdollisen laitealustan hyödyn saavuttamiseksi. Sekä Android, että iOS tukevat Cordovan avulla yhtenäisesti natiiveja toiminnallisuuksia, ja sovelluksessa hyödynnetäänkin yhteystietojen tallennusta, ohjausta soitto-toimintoon, verkkosivuille ja karttapalveluun sekä laitteen omaa tietokantaa tietojen tallentamiseen. Kuviossa 13 on esimerkki yritykselle mahdollisista toiminnoista, jotka avautuvat toimintovalikkoon.



Kuvio 13. Yrityksen toimintojen aukeaminen toimintovalikkoon

Toimintovalikko on yksi Ionic Frameworkin komponenteista ja sen sisällä olevat toimintovaihtoehdot muodostetaan määrätyn muotoisen taulukon avulla. Esimerkkikoodissa 5 esitellään yksinkertaistettuna toimintovalikon ja natiivien toimintojen kutsujen toteutus.

```
let c = this;

let actions = [{
    text: 'Lisää yhteystietoihin',
    icon: 'person-add',
    handler: () =>
        navigator.contacts
            .create({"displayName": c.company.name})
            .save()
}, {
    text: 'Soita',
    icon: 'call',
    handler: () =>
        window.open(`tel:${c.company.number}`,
            '_system',
            'location=yes')
}, {
    text: 'Siirry verkkosivulle',
    icon: 'globe',
    handler: () => {
        c.platform.ready().then(() => {
            cordova.InAppBrowser.open(
                encodeURI(c.company.homePage),
                '_system',
                'location=yes'
            );
        });
    }
}];
```

Esimerkkikoodi 5. Kuvion 13 toimintovalikon yksinkertaistettu ohjelmakoodi

Koodissa actions-taulukko sisältää kuviossa 12 näkyvät kolme ensimmäistä toimintoa taulukon alkioina. Kuvion karttanäkymätoiminto on lähes identtinen verkkosivulle siirtymisen kanssa, joten sitä ei esitellä erikseen esimerkkikoodissa 4. Actions-taulukon alkioille on kullekin asetettavissa teksti (text), ikoni (icon) sekä käsittelijäfunktio (handler). Käsittelijäfunktio suoritetaan vasta käyttäjän valitessa jonkin toiminnon. Yrityksen tiedot on aiemmassa vaiheessa sovellusta asetettu komponentin sisäiseen kontekstiin, jolloin

asettamalla konteksti c-nimiseen muuttujaan rivillä 1 voidaan eri kontekstin funktioissa viitata aiemmin asetettuihin yritystietoihin.

Yhteystietojen lisääminen käyttää aiemmin luvussa 3.2 esitelty navigator.contacts-rajapintaa yhteystiedon luomiseksi ja tallentamiseksi. Rivillä 8 navigator.contacts.create-metodissa voidaan asettaa yhteystiedon kaikki tiedot luotavaan yhteystieto-olioon. Tässä tapauksessa asetetaan vain displayName eli yhteystiedon listalla näkyvä nimi. Seuraavan rivin save-metodi yrittää tallentaa yhteystiedon asynkronisesti, jolloin käytännön toteutuksessa tulee myös varautua toiminnon epäonnistumiseen esimerkiksi näyttämällä virheilmoitus käyttäjälle.

Soittamistoiminto on toteutettu hyvin yksinkertaisesti sovelluksessa ilman Cordovaa. Sovellus avaa käsittelijäfunktiossa selaimen oman window.open-metodin avulla uuden ikkunan, jonka osoitteena annetaan puhelinnumero ja sen eteen tel:-etuliite. Käyttöjärjestelmä tunnistaa tel-protokollatunnisteen ja osaa avata puhelimen oman soittonäkymän ja liittää siihen automaattisesti annetun puhelinnumeron. Metodin kutsussa ”_system”-parametri määrittelee uuden ikkunan avautumaan järjestelmän oletussovelluksen avulla ja ”location=yes”-parametri pakottaa osoiterivin näkyviin, mikäli sellainen on kohdesovelluksessa tuettuna.

Verkkosivulle siirtymiseen käytetään Cordovaa, sillä selaimen window.open-toiminto ei oletusarvoisesti salli useiden verkkosisältöikkunoiden näyttämistä samanaikaisesti Androidin WebView:llä (33). Cordovan avulla voidaan kuitenkin käyttää vastaavaan toiminnallisuuteen InAppBrowser-laajennusta, jota kutsutaan samoilla parametreilla kuin window.open-metodiakin. Myös ”Näytä kartalla” -toiminto on toteutettu käyttäen samaa kutsupohjaa karttapalveluun vievällä osoitteella.

5.3 Käytettävyys ja käyttökokemuksen parantaminen

Verrattuna perinteisiin työpöytäsovelluksiin tai työpöytäkoossa näytettäviin verkkosivuihin mobiilisovellus asettaa pienemmän näytön ja rajoitettujen syöttötoimintojen vuoksi erityisiä haasteita sovelluksen käytettävyyteen. Tässä luvussa on koottuna muutamia sovelluksen kehitysvaiheessa vastaan tulleita käytettävyyshuomioita ja suunnitteluratkaisuja.

Perusperiaatteita käyttäjäkeskeisessä suunnittelussa ovat toimintojen merkityksen selkeys ja niiden tuottama vaste käyttäjälle toimintoja tehtäessä. Sovelluksessa esimerkiksi klikkaamalla alkunäkymässä hakupalkkia hakupalkki siirtyy ruudun ylälaitaan ja tekstinsyöttökenttä aktivoituu paljastaen samalla laitteen oman näppäimistön. Lisäksi näkyviin tulee haettavan tietotyypin valintavälilehdet sekä tyhjää tilaa hakutulosten näyttämiseen. Koko siirtymä on animoitu suoraviivaisesti, jolloin siirtymä tilasta toiseen on selkeä käyttäjälle. Kuviossa 14 havainnollistetaan sovelluksen tilan muutos alkunäkymästä hakutilanteeseen.

Kuvio 14. Hakupalkin siirtymä etusivulta hakutulosnäkymään

Hakutilanteeseen liittyy olennaisesti myös indikaattori haun suorituksesta. Itse haku ja siihen liittyvä tiedonsiirto vievät ennalta määräämättömän verran aikaa, jolloin käyttäjälle on syytä kertoa toimenpiteen olevan suorituksessa. Tähän tarkoitukseen sovelluksessa käytetään animoitua kuvaketta. Selkeät virheilmoitukset haun epäonnistumisesta tai tuloksettomasta hausta ovat tärkeitä luomaan sovelluksesta toimintavarma ja luotettava mielikuva.

Yleisesti sovelluksen tietorakenne ja navigaatio ovat suuressa osassa mobiilisovelluksen kehitystä, sillä tilan säästämisen vuoksi erilaiset navigointiratkaisut ovat usein piilotettuina valikko-napin taakse. Sovelluksen tiedon jäsentelyn ja siirtymien hierarkiatasojen

välillä tulisi olla niin sujuvaa, että käyttäjän tarvitsee käyttää valikkopalkkia mahdollisimman vähän sovelluksen ydintoimintojen suorittamiseen, eli Intellian mobiilisovelluksen tapauksessa yritys- ja yhteystietojen nopeaan löytämiseen.

Sovelluksessa noudatetaan pääsääntöisesti porautuvaa navigointia, missä päätason näkymästä pääsee vain sen alatason näkymiin. Takaisin päätasolle liikutaan taakse-napin avulla. Sovelluksessa on valikkotoiminto, joka näytetään vain päätason sivuilla, jotta sisältö saadaan paremmin esille esimerkiksi yrityksen tarkemmissa tiedoissa.

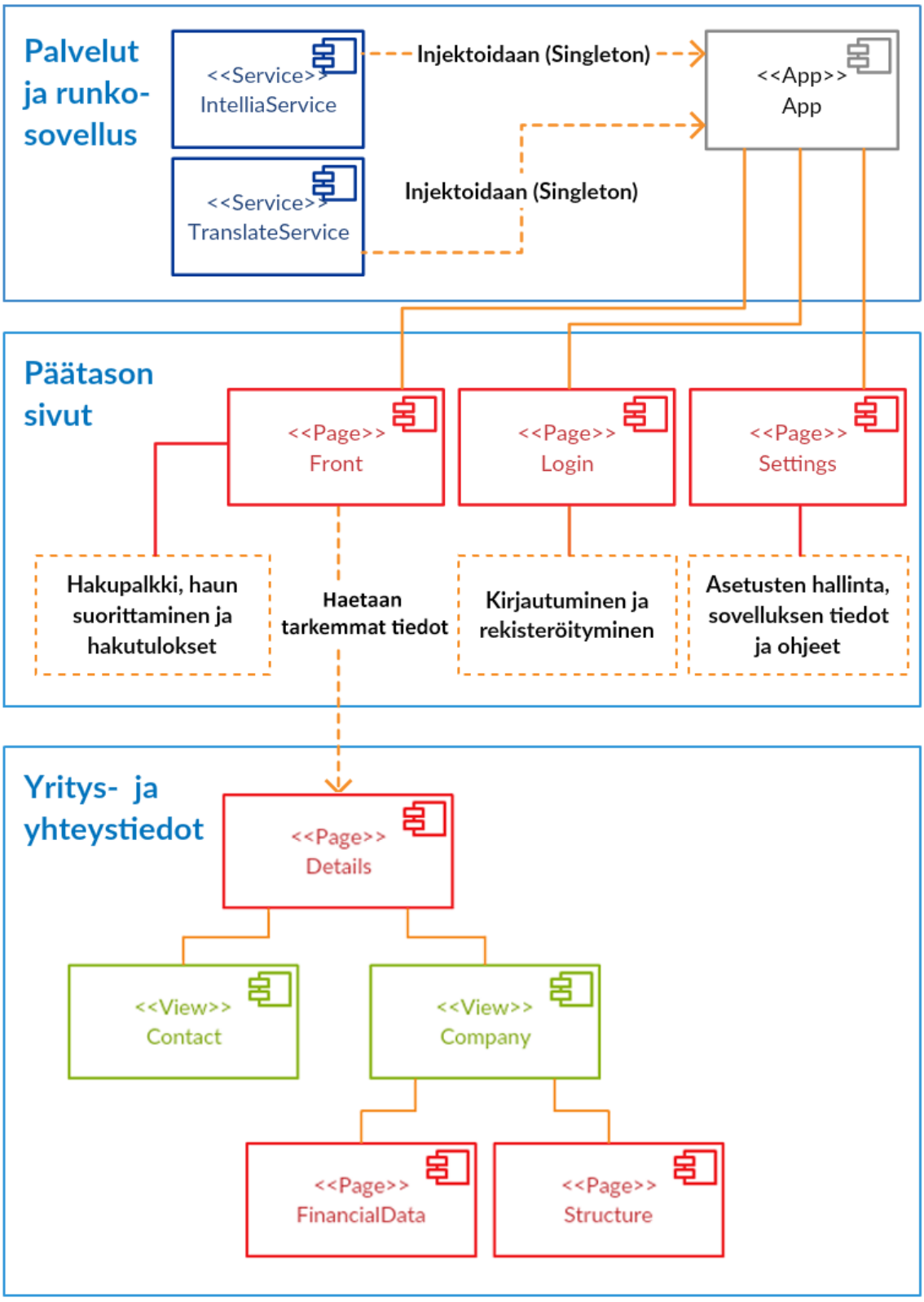
5.4 Lopputuote

Opinnäytetyön loppuvaiheessa sovelluksen ensimmäinen versio on rekisteröitymispalvelun toteuttamista vaille valmiina. Perustoiminnallisuudet toimivat ja ovat testattuja eri kohdelaitteilla. Sovelluskehitys on kuitenkin jatkuva, iteratiivinen prosessi, jolloin muutospyyntöjä ja ominaisuustoiveita tulee vastaan viimeistään ensimmäisten asiakaspalautteiden myötä kirjautumisominaisuuden toteuttamisen jälkeenkin.

Tässä luvussa käydään läpi sovelluksen rakenne komponenttikaaviona sekä esitellään sovelluksen yrityshaun toimintoputki graafisesti. Samalla pohditaan sovelluksen mahdollisia tulevaisuuden kehityssuuntia.

5.4.1 Komponenttikaavio

Sovelluksen tämänhetkinen rakenne noudattaa pitkälti alun perin suunniteltua sivukarttaa, kuitenkin yhdistäen yritysten ja yhteystietojen hakutoiminnallisuuden yhden sivun alle. Perinteisellä luokkamallilla on vaikeaa kuvata järkevästi sovelluksen toimintaa, sillä käytössä on erillinen sovelluskehys. Sen sijaan hieman mukautetun komponenttikaavion avulla sovelluksen toiminta hahmottuu paremmin. Kuviossa 15 esitelläänkin sovelluksen rakenne komponenttitasolla.



Kuvio 15. Sovelluksen komponenttikaavio

Komponenttikaaviossa sovelluksen eri komponentit on esitelty erivärisillä laatikoilla, jotka ovat sidoksissa toisiinsa viivoilla. Näiden lisäksi eri sovelluksen osiot on ympäröity

suuremmalla nelikulmiolla sovelluksen eri tasojen havainnollistamiseksi. Eri värit kuvaavat komponentin tyyppiä, joka mainitaan komponentin sisällä sen ensimmäisellä tekstirivillä. Toisella rivillä kerrotaan komponentin nimi. Viivat kuvaavat komponenttien välisiä toiminnallisuuksia. Yhtenäisellä viivalla kuvataan suoraviivainen edestakainen siirtymä komponenttien välillä, katkoviivalla siirtymään liittyy jokin toiminto. Katkoviivalla korostetuissa laatikoissa kuvaillaan komponentin sisällä suoritettavat erityiset toiminnot ja tarvittaessa niiden sisältö.

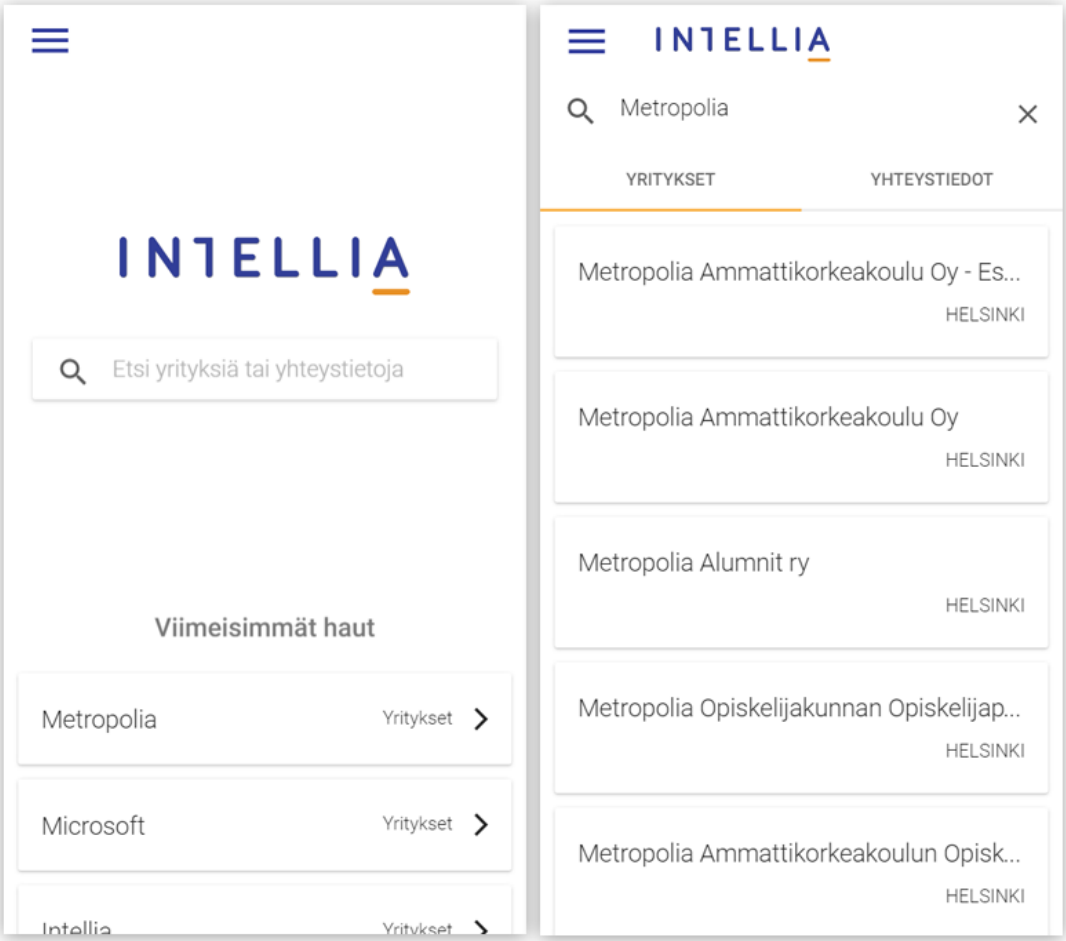
Palvelut ja runkosovellus -osiossa kaksi sovelluksen käyttämää palvelukomponenttia, IntelliaService ja TranslateService injektoidaan Singleton-muotoisina koko muun sovelluksen käytettäviksi. IntelliaService sisältää muun muassa haku- ja kirjautumislogiikan sovellukselle, sekä kaikki tarvittavat rajapintaosoitteet ja sovelluksen asetukset. TranslateService taas on Angular 2 -pohjainen laajennus, jonka avulla sovelluksen eri näkymien tekstit voidaan näyttää eri kielillä yksinkertaisesti.

App-komponentti on sovelluksen runko, jossa alustetaan sovelluksen päätasojen sivut ja niiden hallintakomponentti sekä ohjataan käyttäjä Front-sivukomponenttiin. Front-sivulla käyttäjä voi suorittaa haut sekä yritys- että yhteystietoihin. Käyttäjän valitessa hakutuloksen, siirrytään Details-sivulle.

Details-sivu hakee automaattisesti valitun hakutuloksen tyypin mukaan tarkemmat tiedot hakutuloksesta ja näyttää ne tietotyypin mukaan omassa näkymässään. Yrityshaun puolella käyttäjä pystyy lisäksi navigoimaan yrityksen talous-, henkilöstö- tai organisaatiotietoihin. Molemmilla tietotyypeillä on omat toimintonsa, kuten kohteen tallentaminen yhteystietoihin tai kohteelle soittaminen. Toiminnot määritellään lisäksi saatavissa olevan tiedon mukaan, esimerkiksi yrityksen verkkosivulle ei luoda vierailulinkkiä, mikäli verkkosivulinkkiä ei ole saatavilla.

5.4.2 Yrityshaku käytännössä

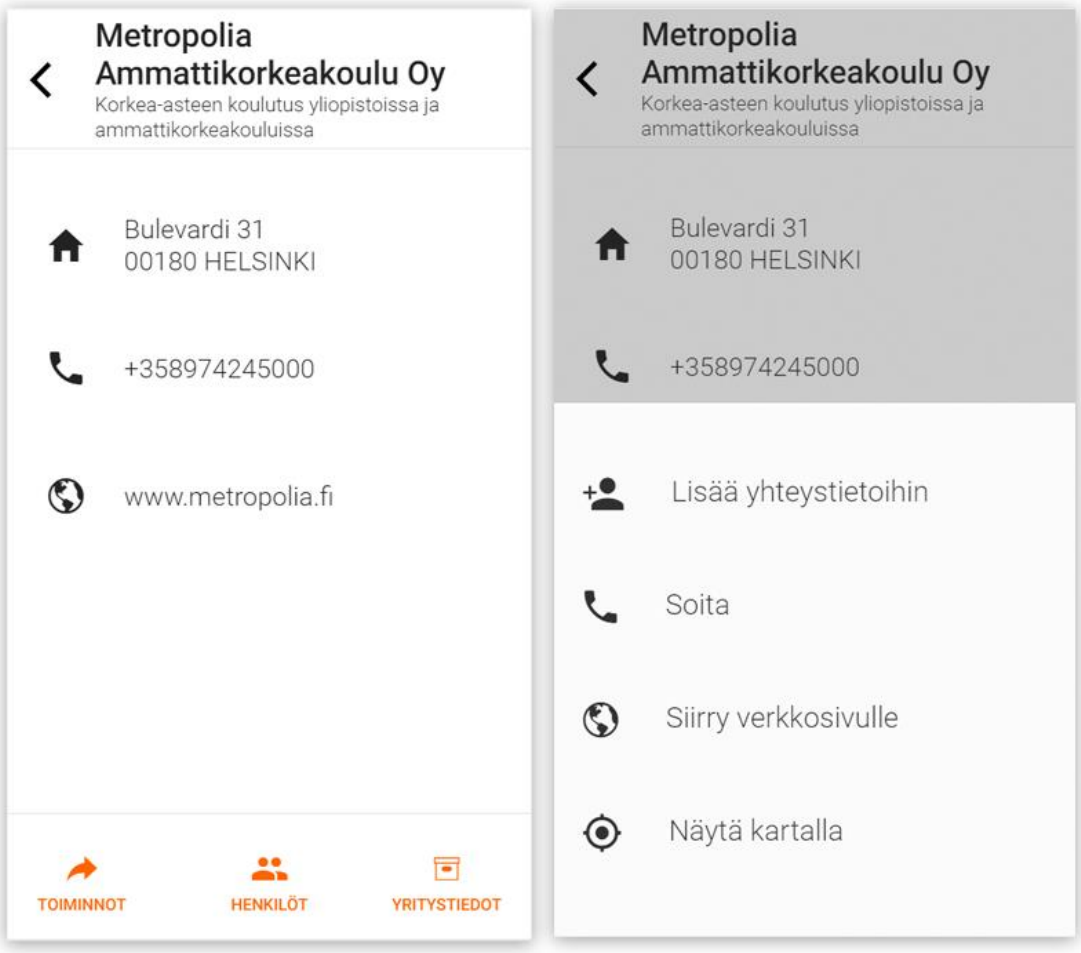
Sovelluksen toiminnasta saa paremman kuvan, kun sen näkee käytännössä. Tämän luvun kuviosarjassa esitellään yrityshaku hakusanalla ”Metropolia” sovelluksen aloitusnäkymästä yrityksen tarkempiin tietoihin.



Kuvio 16. Etusivu ja hakutulosnäkymä

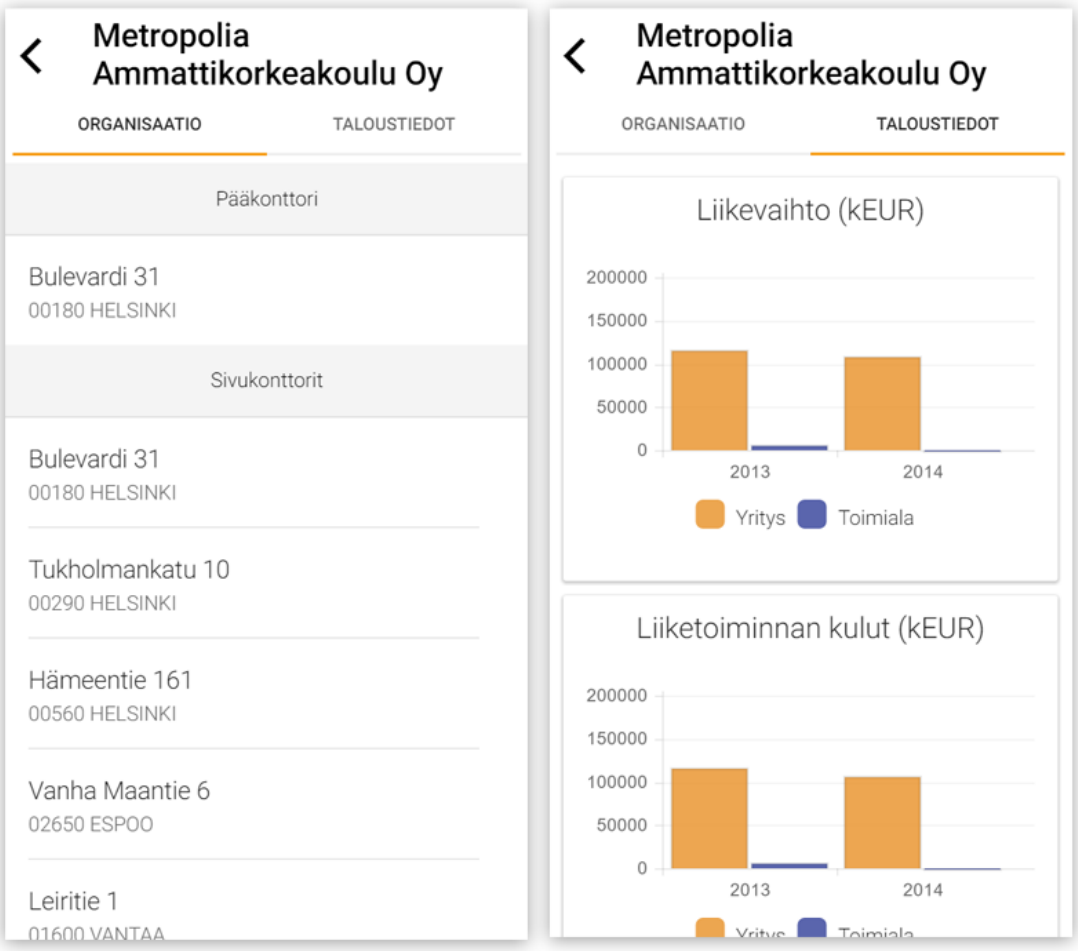
Sovellus käynnistyy kuvion 16 vasemmanpuoleiseen näkymään ja hakupalkkia painamalla siirrytään hakutulosnäkymään. Hakutuloksia haetaan ja näytetään automaattisesti hakusanan muuttuessa kuvan oikeanpuoleisen näkymän mukaisesti. Tässä vaiheessa sovelluksesta voidaan valita hakutulos tarkempaa tarkastelua varten.

Kehitysajatuksena hakutoiminnon lisäksi sovellus voisi tarjota käyttäjälle mahdollisuutta päivittää automaattisesti kohdelaitteen yhteystietoluettelon puhelinnumerot, osoitteet ja muut tiedot uusimmiksi saatavilla oleviksi tiedoiksi. Lisäksi hakutoimintoon voisi liittää helppokäyttöisen suodatintoiminnallisuuden. Tuloksia voisi suodattaa esimerkiksi liikevaihtoluokan tai sijainnin mukaan.



Kuvio 17. Yrityksen perustiedot ja ”Toiminnot”-valikko

Kun hakutulos valitaan, näytetään kuvion 17 vasemmanpuoleisen kaltainen näkymä, jossa luetellaan hakutuloksen perustiedot. Alapalkissa on eriteltyinä yritykseen liittyviä toiminnallisuuksia omina painikkeinaan. Valitsemalla ”Toiminnot”-painikkeen käytettävissä olevat toiminnot näytetään kontekstivalikkona (kuviossa 17 oikealla). ”Henkilöt”-painikkeesta päästään tarkastelemaan yrityksen henkilöstötietoja omaan näkymäänsä. ”Yritystiedot”-painikkeesta päästään tarkastelemaan yrityksen tarkempia tietoja, kuten sen organisaatiorakennetta ja taloustietoja kuvion 18 näkymien mukaisesti.



Kuvio 18. Yrityksen rakenne ja taloustiedot

Kuviossa 18 vasemmalla on yrityksen organisaatiorakenne jaoteltuna konttoreihin, tytäryrityksiin sekä osaomistusyrityksiin. Kohteen osoite on myös mahdollista näyttää kartalla tai kopioida mobiililaitteen leikepöydälle osoitetta napauttamalla. Oikeanpuoleisessa näkymässä voidaan tarkastella yrityksen saatavissa olevia taloustietoja ja vertailla niitä aiempien vuosien tietoihin kuvaajien avulla.

Kehitysajatuksena kuvion 18 näkymään liittyen yrityksen organisaatiorakennetiedot voitaisiin esittää vielä havainnollisemmin esimerkiksi hierarkkisen puunäkymän avulla. Lisäksi rakennetta tulisi voida suodattaa esimerkiksi tekstisuodattimen avulla. Suurissa organisaatioissa tytäryhtiöitä ja sivukonttoreita voi olla useita kymmeniä, jolloin käyttäjän on vaikeaa löytää haluamansa tieto vain listaa selaamalla.

6 Yhteenveto

Mobiilisovelluksen kehittäminen yhdistelee monipuolisesti erilaisia ohjelmistoteknisiä ratkaisuja ja käyttäjäkeskeistä suunnittelua tuoreiden tekniikoiden ja jatkuvasti kehittyvien päätelaitteiden ympärillä. Vaihtoehtoja mobiilisovelluksen kehittämiseksi ovat natiivi-, verkko- tai hybridimobiilisovellus sekä niiden eri variaatiot. Kun halutaan rakentaa eri alustoilla toimiva mobiilisovellus, tehokas vaihtoehto on rakentaa hybridimobiilisovellus. Tässä työssä onkin kuvattu hybridimobiilisovelluksen rakentamisen kehitysprosessi aina sen suunnitteluvaiheesta toteutukseen teknologioiden lisäksi myös käytännön näkökulmasta.

Hybridimobiilisovelluksen rakentamisessa käytetään usein erityistä sovelluskehystä, mikä tehostaa ja helpottaa sovelluksen rakentamista. Työn käytännön sovellusprojektiin valittiinkin Ionic Framework 2 -sovelluskehys, jolla on mahdollista integroida sovellus laitetason toimintoihin. Lisäksi kehys tarjoaa valmiita käyttöliittymäkomponentteja sekä rungon sovelluskehityksen pohjaksi. Kehyksen taustalla on käytetty paljon erilaisia ohjelmiston suunnittelumalleja, jotka edesauttavat selkeän ja laajennettavan koodipohjan rakentamista sovellukseen.

Mobiilisovelluksen rakentamisessa on ohjelmistokehityksen lisäksi otettava kantaa sovelluksen käytettävyyteen. Käytettävyyttä voidaan parantaa esimerkiksi vähentämällä käyttäjältä vaadittavien painalluksien määrää yhdistämällä ja automatisoimalla sovelluksen toimintoja sekä luomalla selkeät siirtymät eri näkymien välille. Lisäksi tunnettujen käyttöliittymän suunnittelumallien käyttäminen sovelluksessa auttaa käyttäjää omaksumaan sovelluksen eri toiminnot paremmin, kun niiden toiminta on entuudestaan tuttua.

Työn lopputuotteena syntynyt Intellian mobiilisovellus on konkreettinen osoitus hybridisovellustekniikan mahdollisuuksista ja soveltuvuudesta käytännön sovelluskehitysprojekteihin. Esiteltyjen tekniikoiden avulla voidaan rakentaa natiivisovelluksien kaltaisesti käyttäytyvä sovellus eri alustoille, yhdellä koodipohjalla. Hybridimobiilisovelluksen rakentaminen lunastaakin sille asetetut odotukset ja sitä on vaikeaa olla suosittelematta uusien mobiilisovellusprojektien kehitysmuotoa pohtiessa.

Lähteet

1 Anand, I., Wasmer D. 2016. Native vs Web vs Hybrid: How to Select the Right Platform for Your Enterprise's Mobile Apps. Verkkodokumentti. <http://www.kinvey.com/native-web-hybrid-developers-ebook>. 2016. Luettu 12.1.2016.

2 Budiu, R. 2013. Mobile: Native Apps, Web Apps, and Hybrid Apps. Verkkodokumentti. <https://www.nngroup.com/articles/mobile-native-apps/>. 14.9.2013. Luettu 10.1.2016.

3 PCMag. 2016. Definition of: native mobile app. Verkkodokumentti. <http://www.pcmag.com/encyclopedia/term/65705/native-mobile-app>. 2016. Luettu 11.1.2016.

4 Apple Inc. 2016. Choosing a Membership. Verkkodokumentti. <https://developer.apple.com/support/compare-memberships/>. 2016. Luettu 11.1.2016

5 Android Developers. 2016. Get Started with Publishing. Verkkodokumentti. <http://developer.android.com/distribute/googleplay/start.html>. 2016. Luettu 12.1.2016.

6 Perez, Sarah. 2015. Consumers Spend 85% Of Time On Smartphones In Apps, But Only 5 Apps See Heavy Use. Verkkodokumentti. <http://techcrunch.com/2015/06/22/consumers-spend-85-of-time-on-smartphones-in-apps-but-only-5-apps-see-heavy-use/>. 22.6.2015. Luettu 15.1.2016.

7 Nielsen Company. 2015. So Many Apps, So Much More Time for Entertainment. Verkkodokumentti. <http://www.nielsen.com/us/en/insights/news/2015/so-many-apps-so-much-more-time-for-entertainment.html>. 11.6.2015. Luettu 18.1.2016.

8 Kwok, J. 2015. Netflix Likes React. Verkkodokumentti. <http://techblog.netflix.com/2015/01/netflix-likes-react.html?m=1>. 28.1.2015. Luettu 18.1.2016.

9 Whitten, B. 2015. What are some examples of good HTML5 mobile apps? Verkkodokumentti. <https://www.designernews.co/comments/62950>. 2015. Luettu 30.1.2016.

10 Moodle. 2016. Moodle Mobile. Verkkodokumentti. <https://docs.moodle.org/30/en/Moodle_Mobile>. 10.2.2016. Luettu 20.2.2016.

11 Visionmobile Ltd. 2014. Developer Economics Q3 2014: State of the Developer Nation. Verkkodokumentti. <http://www.visionmobile.com/product/developer-economics-q3-2014/>. 1.7.2014. Luettu 18.1.2016.

12 Google Developers. 2015. Installable Web Apps with the Web App Manifest in Chrome for Android. Verkkodokumentti. <https://developers.google.com/web/updates/2014/11/Support-for-installable-web-apps-with-webapp-manifest-in-chrome-38-for-Android?hl=en>. 2015. Luettu 18.1.2016.

13 Drifty Co. 2016. All About Ionic. Verkkodokumentti. <http://ionicframework.com/docs/guide/preface.html>. 2015. Luettu 20.1.2016

14 The Apache Software Foundation. 2015. Cordova Documentation Overview. Verkkodokumentti. <https://cordova.apache.org/docs/en/latest/guide/overview/>. 2015. Luettu 15.1.2016.

15 IŞIK, U. 2015. Hybrid Mobile Application Development With Ionic. Verkkodokumentti. <http://www.slideshare.net/UmutIIK/ionic-52121097>. 27.10.2015. Luettu 15.1.2016.

16 Falk, M. 2016. Mobile Frameworks Comparison Chart. Verkkodokumentti <http://mobile-frameworks-comparison-chart.com/>. 29.11.2015. Luettu 18.1.2016.

17 Drifty Co. 2016. Tutorial: Project Structure. Verkkodokumentti. <http://ionicframework.com/docs/v2/getting-started/tutorial/project-structure/>. 2016. Luettu 20.2.2016.

18 The Apache Software Foundation. 2015. Hooks Guide. Verkkodokumentti. <https://cordova.apache.org/docs/en/latest/guide/appdev/hooks/index.html>. 2016. Luettu 20.2.2016.

19 Drifty Co. 2016. Ionic Developer Glossary. Verkkodokumentti. <http://ionicframework.com/docs/v2/resources/what-is/>. 2016. Luettu 20.2.2016.

20 Drifty Co. 2016. Ionic CLI. Verkkodokumentti. <http://ionicframework.com/docs/v2/cli/>. 2016. Luettu 20.2.2016.

21 Bradley, A. 2016. Announcing Ionic Framework 2 Beta. Verkkodokumentti. <http://blog.ionic.io/announcing-ionic-framework-2-beta/>. 10.2.2016. Luettu 10.2016.

22 Drifty Co. 2016. Installing Ionic. Verkkodokumentti. <http://ionicframework.com/docs/v2/getting-started/installation/>. 2016. Luettu 10.2.2016.

23 Drifty Co. 2016. Components. Verkkodokumentti. <http://ionicframework.com/docs/v2/components/>. 2016. Luettu 10.2.2016.

24 Drifty Co. 2016. App. Verkkodokumentti. < http://ionicframework.com/docs/v2/api/decorators/App/>. 2016. Luettu 10.2.2016.

25 Gamma, E. Helm, R., Johnson R., Vlissides J. 1995. Design patterns: elements of reusable object-oriented software. Addison-Wesley 1995.

26 Google. 2016. Angular 2 Glossary. Verkkodokumentti. <https://angular.io/docs/ts/latest/glossary.html>. 2016. Luettu 20.3.2016.

27 Lancina, T. 2016. Ionic 2 Starter Template. Verkkodokumentti. <https://github.com/driftyco/ionic2-starter-tutorial>. 19.2.2016. Luettu 22.2.2016.

28 Drifty Co. 2016. Adding Pages. Verkkodokumentti. <http://ionicframework.com/docs/v2/getting-started/tutorial/adding-pages/>. 2016. Luettu 20.3.2016

29 Microsoft. 2016. The Reactive Extensions for JavaScript. Verkkodokumentti. <https://github.com/Reactive-Extensions/RxJS>. 20.2.2016. Luettu 22.2.2016.

30 Intellia Oy. 2014. Intellia Oy. Verkkodokumentti. <https://intellia.fi/yritys> . 2016. Luettu 20.2.2016.

31 Taloussanomat. 2016. Intellia Oy. Verkkodokumentti. <http://yritys.taloussanomat.fi/y/intellia-oy/helsinki/2438569-7/>. 1.12.2014. Luettu 20.2.2016.

32 Intellia Oy. 2014. Tutustu ratkaisuihimme. Verkkodokumentti. <https://intellia.fi/ratkaisut>. 2014. Luettu 20.2.2016.

33 Android Developers. 2016. WebSettings, Class Overview. Verkkodokumentti. <http://developer.android.com/reference/android/webkit/WebSettings.html#setSupportMultipleWindows(boolean)>. 2016. Luettu 29.3.2016.